

UNIVERSIDADE DE COIMBRA



On the orchestration of operations in flexible manufacturing

Germano Manuel Correia dos Santos Veiga

Doctoral Dissertation, 2009



# On the orchestration of operations in flexible manufacturing

Germano Manuel Correia dos Santos Veiga



Coimbra 2009.



Supervisor: Professor Norberto Pires (Auxiliar Professor at the Faculty of Science and Technology from the University of Coimbra)

Co-Supervisor: Professor Klas Nilsson (Associate Professor at the Department of Computer Science from Lund University)

The candidate performed the experimental work with a doctoral fellowship supported by the European Commission through the project SMERobot, that is an Integrated project funded under the European Union's Sixth Framework Programme (FP6) which also participate with grants to attend in international meetings and for the graphical execution of this thesis. The Faculty of Science and Technology of the University of Coimbra (Portugal) and the Lund University (Sweden) provided the facilities and logistical supports.



*Para ti Mãe...*





# Acknowledgments

---

First of all I would like to thank my supervisors Prof. Norberto Pires, and Prof. Klas Nilsson for their scientific supervision as well as their unconditional support. I can hardly imagine two more complementary researchers working in the same research field, which has provided me with a fruitful experience among robots. To Prof. Norberto, thank you for walking me into industrial robotics, for the patience, example and for the scientific support. To Prof. Klas Nilsson, thank you for providing me with the different perspective, the step back, for all the interesting discussions, and particularly, for the hospitality during my stay in Lund.

To all my colleagues at the Industrial Robotics Laboratory, and especially to Ricardo Araújo, Gabriel Afonso and Pedro Neto, thanks for all the support and help. To all those I've encountered in my industrial experiences, with a special mention to Rui Cancela for his friendship and for all those robotics discussion/dinners, thanks for the industrial grounding.

To all the people that helped me during my stay in Lund, namely, Gorél Hedin, Anders Nilsson, Tjebörn Eckman and Mathias Haäge, thanks for the interesting discussions and for the patience in the opening of my “mechanical” engine to the software world. I would like to thank all the SMERobot team for the debate, discussion and ideas.

The work presented in the thesis has been financially supported by SMERobot, a project funded under the EU's sixth framework programme.

To my friends in general, to my housemates, to Alexandra for the help with the English, to Rui, Carla, Tila, Manuel, Irma and to you Marisa, thanks for your friendship, help and support.

To my father and to my beloved Nini, thanks for all. My achievements are yours too.

Finally, to you Joana, there are no words to express my gratitude. While I'm writing this, you are sleeping next to me because you stated "I won't go home before you are finished". That says it all. Thank you.

# Resumo

---

## **Domínio**

A capacidade de produzir bem é a chave da riqueza. Uma boa produção concretiza-se pela transformação competitiva de matérias-primas em produtos de qualidade para o mercado global. Tal transformação inclui uma série de operações coordenadas de modo a obter a produtividade que permite o aumento da competitividade.

Embora algumas operações possam requerer pessoal especializado, a tendência é para uma crescente automatização. A coordenação das operações automatizadas é também automatizada (através de uma variedade de transportadores, comunicação digital, etc.). Contudo, e embora a produção seja automatizada, a configuração do equipamento é feita manualmente.

## **Situação**

A necessidade de automatização foi inicialmente sentida e aplicada na produção de longas séries, como no caso da indústria automóvel. Com a necessidade de redução de custos e aumento de flexibilidade, a inclusão de máquinas/equipamentos controlados por computador, assim como de interligações por computadores/redes tem sido extensiva. O aumento de software nestes sistemas, as restrições físicas e as (indesejáveis) interconecções lógicas conduzem a um aumento da complexidade, que em empresas de produção em grande escala é obviado por especialistas.

O aumento na procura de produtos personalizados e a rapidez para a sua comercialização determinam a necessidade de uma produção flexível. Contudo, a referida indesejável

complexidade constitui um grande obstáculo para o recurso a soluções (semi)-automatizadas, e postos de trabalho (de alta qualidade) são deslocados para países de mão-de-obra mais barata.

O caso mais difícil prende-se com a utilização de robôs (que é o tipo de máquina mais flexível) nas empresas mais flexíveis, como sejam as Pequenas e Médias Empresas (PMEs). Deste modo, as condições para a utilização de robôs (juntamente com outro tipo de equipamento) em PMEs (incluindo operações manuais e configuração) representam o maior desafio, uma vez que os sistemas técnicos têm de ser estruturados de forma a suportarem a desejada flexibilidade.

À semelhança da incapacidade de uma boa gestão em compensar a falta de competência em actividades como a do comércio ou da investigação científica, as etapas de produção têm de ser apropriadamente construídas e mantidas. Estas últimas representam aquilo em que o produtor se deve concentrar e especializar. Assim, sejam quais forem os avanços que facilitem a produção, as práticas de produção competitiva deverão ser mantidas. Especificamente, processos de trabalho com conhecimentos valiosos não devem ser prejudicados por detalhes técnicos irrelevantes. Novas tecnologias que possam aumentar a competitividade e/ou melhorar as condições de trabalho são obviamente desejáveis. Contudo, estas deverão ser combinadas com princípios de utilização simples para os quais os sistemas de produção são construídos.

### **Tópico**

A complexidade surge tanto na operação como na coordenação, mas também na sua configuração manual. Esta é parcialmente compreensível e gerenciável, uma vez que uma determinada máquina ou processo de fabrico pode ser bastante avançado e complexo. Assim, a complexidade será local. Contudo, e em particular com software envolvido, dependências ad-hoc acidentais entre operações e coordenação tornam a situação actual mais difícil. Adicionalmente, a configuração de cada máquina e subsistema envolve uma variedade de interfaces de utilizador e ferramentas de configuração/programação. Uma vez mais, as grandes empresas podem contar com engenheiros altamente qualificados para lidar com estes problemas, enquanto que nas PMEs a situação fica facilmente ingovernável. Consideramos que podemos referir o problema como uma questão de orquestração.

Orquestração é então definida como o arranjo, coordenação e gestão (semi-) automatizada de complexos sistemas de produção, incluindo as suas interacções em termos de comunicação e os seus serviços em termos de controlo por computador.

O tópico pode então ser formulado como a procura de princípios óptimos, ou pelo menos exequíveis, para a orquestração de processos produtivos de pequena escala. Na realidade, existem aspectos fundamentais envolvidos que não representam apenas uma questão de engenharia a ser solucionada. Ao contrário, a situação requer uma abordagem científica com especial atenção para a recente tecnologia de suporte proveniente de outras áreas.

### **Abordagem científica**

A procura de princípios apropriados a PME's para a orquestração de processos de produção não se presta a uma análise teórica, que por sua vez requer a utilização de modelos formais e derivação das soluções (sub-)óptimas e suas propriedades. Embora tivesse sido preferível obter provas formais de uma solução óptima, a complexidade do equipamento, o envolvimento de humanos, a considerável desorganização das PME's, a necessidade de aderir às práticas industriais, assim como a variedade de PME's existentes, dificultam uma abordagem teórica. Deste modo, foi seguida uma abordagem empírica. A dificultar o teste e a avaliação de uma abordagem empírica estão as possíveis variações no software envolvido, juntamente com a constante mudança que ocorre na produção em sistemas de produção flexíveis. Adicionalmente, a experimentação não pode ser conduzida em ambientes industriais (onde esta iria perturbar a produção). Assim sendo, as experiências deverão ser cuidadosamente seleccionadas e conduzidas em laboratório com recurso a equipamento industrial. Ainda assim, dadas as condições e singularidade de algum equipamento, não é fácil reproduzir os resultados noutros locais, o que constitui um problema para a validação e aceitação dos mesmos. Contudo, caso a solução sugerida em termos de princípios técnicos possa ser encontrada ou confirmada por resultados publicados de investigação independente, ou se técnicas relacionadas derem origem a novos produtos, tal pode desejavelmente contribuir para a validação de resultados. Isto é, embora resultados semelhantes para outros laboratórios sejam

válidos, as diferenças actuais vão revelar a existência de variações que merecem estudo detalhado.

### **Abordagem técnica**

Os sistemas de produção flexíveis consistem em equipamento distribuído do ponto de vista computacional. Tipicamente, os diferentes aparelhos não foram concebidos para operarem em conjunto, mas devem, no entanto, ser de fácil configuração no local de trabalho. Uma abordagem básica passaria pela utilização de plataformas de software que suportassem componentes distribuídos de uma forma flexível. Contudo, as plataformas computacionais existentes podem não satisfazer com eficiência a necessidade do equipamento integrado, podendo revelar falta de robustez, nomeadamente nas interconecções. Deste modo, é necessário combinar com algum cuidado a tecnologia existente e confrontar as soluções sugeridas com as necessidades actuais das empresas.

A abordagem seguida encontra-se dividida em quatro partes:

1. Suportar o acoplamento fraco entre componentes de forma a obter simples composição quando o equipamento é instalado ou substituído. As interacções necessitam ser assíncronas e baseadas em eventos através de interfaces bem definidas e auto-explicativas, contendo serviços definidos em termos de operações de produção (e não em termos de software interno).
2. Produzir princípios unificados para interacção com utilizador e interfaces, permitindo que utilizadores não especializados possam (re)configurar e (re)programar o sistema de produção. Uma interacção com o utilizador que permita combinar operações básicas que resultem num novo serviço, o qual deverá idealmente estar facilmente acessível através de interfaces programadas e manuais.
3. As abordagens baseadas em modelos têm-se revelado eficazes para desempenho e reutilização. No entanto, os modelos consistem em elevado nível de conhecimento e são dispendiosos de obter no âmbito da flexibilidade e desorganização das PME's. Uma melhor abordagem é permitir a visibilidade do conhecimento envolvido numa determinada etapa

em termos do processo de produção, de modo a que o operador transmita inteligência através de uma interface simples.

4. O software é por defeito não descritivo, assim como a execução sequencial de código imperativo não se compõe. Conhecimento no metanível e descrições declarativas deverão ser utilizadas, se possível, sem comprometimento dos itens anteriores. O objectivo é gerar software ao nível de aplicação, partindo de descrições de alto nível.

A avaliação experimental deverá verificar técnicas individuais como tal, e os resultados deverão ser comparados com investigação relacionada. A abordagem global consiste em combinar os resultados das diferentes partes em princípios aplicáveis a futuros processos de produção das PMEs.

## **Resultados**

O uso de arquitecturas orientadas a serviços (SOA) nas redes empresariais resolveu as limitações das arquitecturas orientadas a componentes no que diz respeito ao acoplamento através da standarização das interfaces, protocolos de comunicação, gestão de transacções, e segurança, entre outros. SOA ao nível do dispositivo é o resultado da importação de princípios SOA para os sistemas embebidos com algumas importantes diferenças, nomeadamente: inclusão de padrões de mensagens *publish/subscribe*, descoberta e descrição directa entre dispositivos, e modelos descritivos genéricos. Numa primeira fase, este trabalho validou os inúmeros trabalhos realizados sobre a aplicação de SOA ao nível do dispositivo em ambiente industrial com o teste num protótipo de célula de trabalho. De seguida foi levado a cabo um trabalho de avaliação comparativa entre duas SOA ao nível do dispositivo com estilos de arquitectura diferentes, servindo como base aos restantes desenvolvimentos da tese. Ainda que os resultados desta avaliação tenham mostrado o grande avanço proporcionado pelo uso de SOA, nomeadamente no que diz respeito ao desacoplamento entre componentes atingido, alguns aspectos críticos para o seu uso efectivo ainda estavam por resolver, designadamente:

1. A geração e a especificação dos serviços ao nível da tarefa
2. A definição de uma linguagem de orquestração adequada às SOA ao nível do dispositivo.

Uma abordagem baseada em tarefas, quando relacionadas com processos de manufactura, consubstancia-se na capacidade de disponibilizar um mecanismo flexível (e amigável para o utilizador de uma PME) para a especificação das interfaces de rede. Os programas de robô são um elemento chave na flexibilidade do robô e este trabalho mostrou que o seu uso para a definição de interfaces vai elevar a fasquia da flexibilidade para o nível das interligações. A natureza procedimental de muitas linguagens de robô encaixa-se perfeitamente com o padrão de mensagens definido nas plataformas SOA, com uma mistura de variáveis de estado definidas a partir de variáveis do robô, e com acções definidas a partir de métodos da linguagem robô.

A definição de uma linguagem de orquestração preencheu uma lacuna nos padrões de orquestração: sistemas conduzidos a eventos. Estes sistemas definem estados e transições de uma forma clara, potenciando a capacidade do utilizador de acompanhar o estado do sistema. *Statecharts* constituem um par adequado para a arquitectura SOA, uma vez que as transições de estado são baseadas em eventos, que no nosso caso são eventos na rede, mas os estados (e também as transições) incluem igualmente acções, que podem ser mapeadas para operações.

A avaliação empírica efectuada mostrou uma previsível boa curva de aprendizagem para estes sistemas, em parte devido às vantagens associadas à sua semelhança a técnicas de automação tradicionais, como os *Sequential Function Charts*. Os resultados desta avaliação são positivos e justificam esforços suplementares para efectuar testes em aplicações reais, o que neste caso implica utilizadores de PME reais.

## **Conclusões**

Três conclusões devem ser retiradas desta tese:

A estratégia proposta para a especificação de serviços é um elemento chave no futuro do uso de SOA ao nível dos dispositivos, devido à importância da definição das interfaces no sucesso destas arquitecturas. A programação ao nível da tarefa é desta forma transferida da programação dos robôs para o nível da rede.

Uma linguagem conduzida a eventos foi definida para a orquestração. Testes revelaram o seu uso e compatibilidade com as necessidades das células de fabrico das pequenas e médias



empresas, nomeadamente estados explícitos e transições baseadas em eventos. Esta abordagem preenche uma lacuna nos padrões de orquestração existentes na indústria e constitui uma excelente base de trabalho para o futuro.

Finalmente, foram abordadas técnicas baseadas em conhecimento, e avaliada a sua integração com a arquitectura definida anteriormente. Estes estudos mostraram a importância das estratégias descritivas e as inúmeras possibilidades abertas quando a semântica é adicionada aos sistemas industriais baseados em software, especialmente quando suportados em bem estabelecidas tecnologias de rede, como as descritas anteriormente.



# Abstract

---

Our ability to manufacture well is the key to our wealth. Obtaining a wider range of different (and better) products in a sustainable way in terms of labour and environment is the big challenge faced by modern manufacturing. In the last few decades, automation has played a key role in the enhanced productivity of mass-production industries, but there has been a paradigm shift: global consumers ask for customization, leading manufacturers to target mass customization and consequently requiring new levels of flexibility for automation. An industrial robot is usually considered to be a flexible machine, which is only true within the large plant scenario. Small enterprises, which are by nature the most flexible ones, do not make use of robot systems as they could, because robot flexibility, which relies on reprogramming and reconfiguring, cannot be taken on by the SME (Small Medium Enterprises) worker at the workshop, and hiring specialists is unacceptable in terms of costs. The easy reconfiguration of a robotic work-cell, which is a distributed environment with computation in different platforms that are coordinated by software, is hindered greatly by the dependencies between cell components. This thesis addresses the problem of dependencies by proposing principles and mechanisms for the orchestration of complex manufacturing systems, i.e., the (semi-) automated coordination of their interactions in terms of communications and computer control.

The industrial environment, especially regarding robotics, does not lend itself to theoretical analysis due to the amount of work needed to reach formal models. This is exacerbated in the unstructured SME environment and when working with user-in-the-loop systems. Therefore,

the approach used in this thesis was mainly empirical, with validation through laboratory prototypes used by some representative users.

The approach consisted of the following parts:

1. supporting loose coupling between components to promote simple composition of services to enable an easier reconfiguration;
2. defining unifying principles in terms of user interaction, by taking into account current robot technologies and improvements from other scientific areas, namely enterprise level networking;
3. enabling a task-based view of knowledge in terms of the manufacturing processes to promote the reconfiguration of the system by process rather than robotics specialists;
4. proposing declarative techniques that support easy configuration of the work-cell in terms understandable by the SME user.

The use of service-oriented architectures (SOAs) in the business world has tackled the limitations of component-oriented architectures in terms of coupling through the standardization of interfaces, communication protocols, transaction management, and security, among others. Device-level SOA are the result of the porting of SOA principles to the embedded level with the addition of several important features, namely: publish/subscribe messaging patterns, peer-to-peer discovery description and generic templates. At an initial stage, this work has validated the current trend of using device-level SOA in industrial environments by testing their use against a prototype work-cell. Following this, a comparison was made between device-level SOA platforms that embodied two different architectural styles. This served as a basis for the rest of the thesis. Although the results from this evaluation have shown the great advances brought about by the use of device-level SOA, for instance in terms of decoupling, some critical issues for their effective use remained unchallenged, namely:

1. The generation and the specification of task-level services (contracts), with transparent and easy-to-use techniques for the SME user.

2. The definition of orchestration techniques adapted to the device-level SOA, with adequate expressiveness and simplicity for the SME user.

A task-based view of knowledge in terms of manufacturing processes relies on the ability to provide a flexible (SME user-friendly) mechanism for the specification of network interfaces. Robot programs are the key element for robot flexibility. This work has shown that their use in interface definition will feature flexibility also at the interconnection level. The procedural nature of many robot programming languages copes perfectly with the device-level SOA messaging style, with a mixture of evented state variables defined by robot variables, and with actions defined by robot methods.

The definition of an orchestration language has addressed a missing pattern in service orchestration: event-driven systems. These systems define states and transitions in a clear way, thus enhancing the user's ability to predict the state of the system. Statecharts provide a perfect match for the device-level SOA since the state-transitions are event-based, but the states (and the transitions) embody actions, which can be mapped to operations. The empiric evaluation made with several types of users has shown the predictable steep learning curve of these systems, partly due to their resemblance to traditional automation techniques like SFCs. The results are therefore positive and justify the effort of testing the system against real applications, which in this case means with real SME users.

Descriptive techniques for software are one of the keys to establish the bridge between humans and computer programs. Despite recent evolutions, these techniques are not yet ready for use, and supportive technologies and methodologies need to be tested. In this work, a cell specification language has been defined and software developed that configures a programming-by-demonstration robotic work-cell.

In conclusion, there are three main outcomes from this thesis.

The proposed strategy for the specification of services is a key enabler in the future use of device-level SOA in industrial robotics, due to the importance of interface definition in the

success of these architectures. Task-level programming is in this way extended from the robot programming level to networked devices.

An event-driven language has been defined for the orchestration. Tests revealed its ease of use and compatibility with the orchestration needs of SME work-cells, namely: explicit states and event-based transitions. This approach fulfils a missing orchestration pattern in the industry, and provides an excellent basis for future work.

A descriptive robotic cell specification has been introduced that supports the automatic work-cell (re)configuration. This work has shown the importance of descriptive knowledge in automation, especially when supported by the networking integration techniques described previously.

# Figure list

---

FIGURE 1 SWITCHED ETHERNET AND COMMUNICATION BUFFERS .....	12
FIGURE 2. PROFIBUS-IRT CHANNEL DIVISION [22] .....	15
FIGURE 3. ETHERCAT'S TOPOLOGY AND CHANNEL DIVISION [48].....	15
FIGURE 4. SERVICE ORIENTED ARCHITECTURE .....	25
FIGURE 5. CLIENT-SERVER, STATELESS, CACHEABLE PROTOCOL (ADAPTED FROM [59]) .....	28
FIGURE 6. REST ARCHITECTURE (ADAPTED FROM [44]) .....	29
FIGURE 7. DIFFERENT STEPS OF AN UPNP NETWORK .....	40
FIGURE 8. DSSP SERVICE STRUCTURE (ADAPTED FROM [85]) .....	47
FIGURE 9. JOIN ARBITER.....	50
FIGURE 10. OPC UA ARCHITECTURE- ADAPTED FROM [78].....	51
FIGURE 11 EXPERIMENTAL SETUP AT THE LABORATORY. ....	54
FIGURE 12. GENERAL UPNP ARCHITECTURE .....	55
FIGURE 13. UPNP DEVICE FOR THE CONVEYOR.....	56
FIGURE 14. UPNP DEVICE DEVELOPED FOR THE INDUSTRIAL ROBOT .....	57
FIGURE 15. UPNP DEVICE DEVELOPED FOR THE SMART CAMERA AND GRAPHICAL INTERFACE .....	58
FIGURE 16. UPNP DEVICE DEVELOPED FOR THE VOICE INTERFACE. ....	59
FIGURE 17. UPNP CONTROL POINT: CELL PROGRAMMER INTERFACE.....	60
FIGURE 18. ABBIRC5 CONTRACT .....	62
FIGURE 19. ABBIRC5 XSLT/HTML INTERFACE .....	63
FIGURE 20. ORCHESTRATION USING MICROSOFT ROBOTICS STUDIO.....	64
FIGURE 21. GENERIC UPNP CONTROL POINT .....	66
FIGURE 22. PDA GENERIC CONTROL POINT .....	67
FIGURE 23. GENERAL ARCHITECTURE .....	73
FIGURE 24. ABB'S RAPID PROGRAM STRUCTURE (ADAPTED FROM [93]) .....	79
FIGURE 25. SOFTWARE ARCHITECTURE .....	80
FIGURE 26. GRAPHICAL INTERFACE OF THE RAPID SERVICE GENERATOR .....	84

FIGURE 27. HISTORICAL PROGRESS OF WORD ERROR RATES FROM SPEAKER INDEPENDENT SPEECH RECOGNITION FOR INCREASINGLY DIFFICULT SPEECH DATA (ADAPTED FROM [120]) .....	89
FIGURE 28. VOICE RECOGNITION INTERFACE (SAPI 5.1) .....	93
FIGURE 29. MICROWAVE OVEN.....	104
FIGURE 30. PDA INTERFACE.....	107
FIGURE 31. STATECHART WITH CONTROL VIA VOICE OR PDA .....	107
FIGURE 32. GENERAL ARCHITECTURE - CELL PROGRAMMING.....	108
FIGURE 33. CELL PROGRAMMER .....	109
FIGURE 34. STATE MACHINE COMPOSITION CONTEXT MENU .....	110
FIGURE 35. PROTOTYPE NEW VERSION OF THE CELL PROGRAMMER .....	112
FIGURE 36. 3D TRAJECTORY GENERATOR UPnP SERVICE.....	113
FIGURE 37. AUTOMATIC GENERATION OF SPEECH GRAMMAR AND ROBOT CODE .....	118
FIGURE 38. FULL CODE GENERATION FOR PBD WORK-CELL.....	124
FIGURE 39. VOICE RECOGNITION SOFTWARE WITH AUTOMATIC GENERATED GRAMMAR. ....	126



# Code listings

---

LISTING 1. NOTIFICATION MESSAGE .....	41
LISTING 2. XML UPNP DEVICE DESCRIPTION MESSAGE – DEVICE PROPERTIES .....	41
LISTING 3.XML UPNP DEVICE DESCRIPTION MESSAGE – SERVICE LISTING .....	42
LISTING 4. MESSAGE FOR UPNP METHOD CALLING .....	42
LISTING 5. SUBSCRIPTION MESSAGE .....	43
LISTING 6. DSSP STATE .....	47
LISTING 7. ABBIRC5 STATE.....	62
LISTING 8.WSDL SAMPLE CODE FOR A SIMPLE IDENTIFICATION SERVICE .....	76
LISTING 9. C# CLASS THAT IMPLEMENTS A SIMPLE IDENTIFICATION SERVICE .....	77
LISTING 10. SAMPLE ROBOT PROGRAM WITH SERVICE ATTRIBUTES .....	82
LISTING 11. GENERATED SOCKET SERVER - PART A .....	85
LISTING 12. GENERATED ABB RAPID TCP/IP SOCKET SERVER - PART B .....	86
LISTING 13. SAMPLE SRGS GRAMMAR. ....	92
LISTING 14. RECOGNITION HANDLING DELEGATE: RETRIEVING SEMANTIC PROPERTIES .....	95
LISTING 15. EXTRA PORTTYPE AND MESSAGES REQUIRED FOR THE CONSUMER INPUT .....	99
LISTING 16. PARTNER LINK AND VARIABLE DEFINITION FOR BPEL4WS.....	99
LISTING 17. BPEL4WS SEQUENCE DEFINITION .....	100
LISTING 18. SCXML SAMPLE SPECIFICATION.....	105
LISTING 19. XML PROGRAM FOR THE CONVEYOR CELL.....	106
LISTING 20. CONVEYOR START EVENT SPECIFICATION .....	110
LISTING 21. CELL SPECIFICATION EXAMPLE.....	117
LISTING 22. ABSTRACT GRAMMAR FOR CELL SPECIFICATION .....	120
LISTING 23. ASPECT FOR THE CODE GENERATION OF THE I/O ROBOT CONFIGURATION FILE.....	121
LISTING 24. JRAG FOR BINDING DECLARATIONS WITH THEIR USES .....	122
LISTING 25. JRAG WITH NON-TERMINAL ATTRIBUTES. ....	123
LISTING 26. VOICE COMMANDS .....	124

LISTING 27. GENERATED SPEECH RECOGNITION GRAMMAR .....	124
LISTING 28. GENERATED ROBOT CODE .....	125
LISTING 29. GENERATED ORCHESTRATION .....	125

# Table list

---

TABLE 1 AVERAGE ANNUAL CHANGE IN LABOUR PRODUCTIVITY (IN %), IN NON-PRIMARY ENTERPRISE BY SIZE-CLASS, EUROPE-19 1988-2001 (ADAPTED FROM [1]).....2

TABLE 2 CLASSIFICATION OF SOLUTIONS (ADAPTED FROM [4]) .....11

TABLE 3 EFFECTS ON THE USE OF SERVICE ORIENTED ARCHITECTURES IN AUTOMATION [76] .....36

TABLE 4. STATE VARIABLES FOR THE UPNP PLAYCD TEMPLATE (ADAPTED FROM [80]) .....44

TABLE 5. UPNP AND DPWS PROTOCOL COMPARISON.....45



# Contents

---

1	Introduction .....	1
1.1	Problem.....	4
1.2	Approach.....	5
1.2.1	Scientific approach.....	5
1.2.2	Technical approach .....	6
1.3	Organization of the thesis.....	7
1.4	Publications.....	8
2	Background .....	11
2.1	Predictability and safety .....	11
2.1.1	Ethernet networking and real time .....	11
2.1.2	PC-based control, programming languages.....	16
2.2	Robotic work-cell .....	17
2.2.1	Human/robot interaction .....	18
2.2.2	Reconfiguration of a robotic work-cell .....	19
2.2.3	Industrial robotic work-cell programming.....	20
2.2.4	Holonic cell structure .....	21
2.3	Middleware .....	21

2.4 Service oriented architectures.....	22
2.4.1 WS* - Web Services .....	26
2.4.2 REST Web Services .....	28
2.5 Reactive systems.....	30
2.6 Messaging patterns.....	31
2.6.1 Remote Procedural Calling RPC .....	31
2.6.2 Asynchronous RPC evolutions .....	32
2.6.3 Publisher/subscriber .....	32
2.7 Device-level SOA .....	33
2.7.1 Industrial use of device-level SOA .....	35
2.8 Compiler technologies .....	37
2.9 Final note .....	38
3 SOA Middleware evaluation .....	39
3.1 Universal Plug-n-Play .....	39
3.1.1 UPnP device architecture.....	39
3.1.2 UPnP Device Control Protocols (UPnP – DCP) .....	43
3.2 Device Profile for <i>WebServices</i> (DPWS) .....	44
3.3 Decentralized Services Structure Protocol (DSSP) .....	46
3.3.1 DSSP service state .....	47
3.3.2 Ports, handlers and events .....	48
3.3.3 Abstraction, contracts, partnership and manifests .....	48
3.3.4 MSRS middleware .....	49
3.4 OLE for Process Control – Unified Architecture (OPC-UA) .....	50
3.5 Platform choice .....	52

3.6 Case study- test-bed description .....	53
3.7 UPnP setup.....	55
3.8 DSSP setup .....	60
3.9 UPnP/DSSP comparison and conclusions .....	64
3.9.1 Architectural style comparison.....	65
3.9.2 Platform comparison .....	67
3.9.3 Conclusions .....	70
4 Proposed solution .....	71
5 Service generation .....	75
5.1 Contracts in SOA .....	75
5.2 Industrial robot programs.....	77
5.2.1 Robot programming languages.....	78
5.2.2 ABB's Rapid language.....	78
5.2.3 Software developed.....	80
5.2.4 Extensions to Rapid language .....	81
5.2.5 Conclusions and future work .....	87
5.3 Speech recognition .....	87
5.3.1 Speech platforms .....	90
5.3.2 Speech for Industrial automation .....	91
5.3.3 Speech recognition grammars .....	91
5.3.4 Voice recognition integration in robotic work-cells .....	93
5.3.5 Conclusions .....	96
6 Robotic work cell programming.....	97
6.1 WS*- Web Services Composition.....	97

6.1.1 Business Process Execution Language for Web Services.....	98
6.1.2 Semantic web services and ontology.....	101
6.2 Visual Programming Languages in industrial automation .....	101
6.3 Analysis - Device-Level service composition.....	102
6.4 StateCharts.....	103
6.4.1 Statecharts eXtensible Markup Language.....	104
6.5 Proposed language and tests.....	105
6.5.1 Software developed for service orchestration .....	108
6.5.2 Evaluation and improved versions.....	110
6.5.3 Other Tests.....	112
7 Automatic configuration .....	115
7.1 Configuration of PbD cells.....	115
7.2 Cell functionality specification.....	116
7.3 Software developed .....	119
7.3.1 JastAdd .....	119
7.4 Integration in the overall service oriented system.....	123
7.4.1 Demonstration software.....	125
7.5 Conclusions .....	126
8 Conclusions .....	129
8.1 SOA evaluation.....	129
8.1.1 Results.....	130
8.1.2 Future.....	131
8.2 Service generation .....	131
8.2.1 Service generation results .....	132



8.2.2 Future.....	132
8.3 Orchestration of services .....	133
8.3.1 Results.....	133
8.3.2 Future.....	134
8.4 Automatic configuration of work-cells .....	134
8.4.1 Future.....	135
8.5 Final remarks.....	135
Bibliography .....	137
Appendix A - Abbreviations .....	151
Appendix B – Video List .....	153



# 1 Introduction

---

In 2000, the European Union (EU) approved the Lisbon Strategy [1], a strategic development plan that aims to make the EU the *most dynamic and competitive knowledge based economy*. The main objective of the Lisbon Agenda is to *deal with the low productivity and stagnation of economic growth in the EU*.

Looking into **European demographics**, we can see that the workforce over the next few years will be increasingly older and, according to the Lisbon Strategy, some of this workforce will need to be employed in **manufacturing**.

Manufacturing has a vital role to play in a sustainable economy, taking fully into account the technical, environmental and social dimensions. The services sector (wealth consumer) has become global, boosting the challenges of manufacturing (wealth producer) in terms of flexibility. Consumers with global knowledge ask for more personalized and complex products, with high standards both in terms of product quality and manufacturing processes. Good manufacturing means competitive transformation of resources into quality products that can be sold on the nowadays global market, while keeping labour issues, environmental concerns and sustainability in mind. This transformation includes a number of operations, such as skills, knowledge, tools and equipment, that are efficiently coordinated according to manufacturing practices, in order to accomplish the kind of productivity that enables competitiveness.

Considering data from 1988 to 2001 [2](Table 1), it is notorious that the increase rate of labour productivity is always substantially smaller in Small Medium Enterprises (SMEs) than in Large

Scaled Enterprises (LSEs). Industrial automation is a key element to increase productivity, but its use in smaller companies is more difficult. Current automation technologies have been developed for high volume companies, and their use in SMEs results in what has been called *the automation trap* [3]: SMEs rely on low wages or use automation technologies designed for other scenarios.

**Table 1 Average annual change in labour productivity (in %), in non-primary enterprise by size-class, Europe-19 1988-2001 (Adapted from [1])**

Productivity	1998/1990	1990/1993	1993/2001	1998/2001
SMEs Micro	1.6	1.0	1.4	1.4
SMEs Small	1.7	2.1	1.8	1.9
SMEs Medium	1.6	3.0	2.2	2.3
SMEs Total	1.6	1.8	1.7	1.7
LSEs	2.4	3.0	2.7	2.8
All enterprises	1.9	2.3	2.2	2.2

Successful manufacturers should be able to concentrate on developing competitive manufacturing processes. This is what they are good at and responsible for. New technology should be able to enhance their activities without compromising the valuable process knowledge of their workforce.

New technologies that can improve competitiveness and/or working conditions are of course highly desirable, but they need to be combined with easy-to-use principles for how manufacturing systems are built.

The gap between current automation technologies and SME needs, that induces the automation trap, applies not only to single automated operations but also to the coordination of these operations, i.e., the management of both material and information flow. The increasing amount of computer-controlled equipment, as well as network-based interconnections, has increased the amount of software in these systems. This leads to **physical**

**restrictions** and undesired **logical interconnections** that are no longer apparent or easily understood. Hence, the complexity increases. In large-scale manufacturing, this increase in complexity is managed by specialists.

With the demand on flexibility, the above-described undesirable complexity poses a major obstacle for semi-automated and automated solutions. Some workplaces may relocate to low labour-cost countries as a solution. . The most difficult case has to do with the most flexible type of company, the SME, when trying to use the most flexible type of machine, **the robot**. Thus, the conditions for the use of robots and other automated equipment in manufacturing SMEs represent the most challenging situation, since the technical systems need to be structured in a way that supports the desired flexibility.

Industrial robotic arms are now a mature technology, especially in terms of mechanics, electronics and programming. For several years, it was expected by the majority of players that the robotics market should expand to other types of industries. However, recent numbers show that automotive industries and their suppliers still account for around 60% of the total annual sales of robots [2]. There are two main reasons for this fact: **cost** and **flexibility**.

Current robot technologies need high volume products and fixed installations to provide the desired **reduction of cost** and increase of productivity. Their use in SMEs results in important new costs: the **hiring of technicians exclusively to program robots**; more frequent reprogramming because of small-size batches; extensive **reconfiguration of tools and devices that complement the robot** in order to achieve functionality in the variety of tasks and products required; and more devices. These new costs are the real barrier in the adoption of robots in SMEs not the initial capital outlay for the robot, since the price of an industrial robot has fallen to 25 percent of its price in 1990 (quality adjusted) [2].

From a machinery point of view, robots are normally classified as **flexible automation**, but this classification is only valid for large companies. The flexible automation concept relies on the ability of industrial robots to be reprogrammed and reconfigured, but these tasks are still performed by specialized technicians and automation experts. Looking at SMEs, it is easy to find many features missing from modern industrial robots for them to be co-workers that help

humans in harder tasks. An SME shop floor is unstructured and changes occur very often in the layout, which demands robotic systems to be **easier to reconfigure**. Small batches require frequent reprogramming, demanding **easy-teaching techniques**. An SME-enabled robot should interact closely with humans, with work interleaving and without barriers. This requires **advanced interaction with humans** and **rich environment awareness**. All these advances are minimalist mimics of basic characteristics of the human worker.

To achieve this level of flexibility, there are many aspects of robot technology that need to be improved, namely [3]:

1. The safe human-aware space-sharing robot.
2. The robot capable of understanding human-like instructions.
3. Mechanisms and architecture to plug-n-produce.
4. Robot task generation based on product/process data.

Some of these aspects will be addressed in this work, with special emphasis on point 3, although providing some knowledge regarding how technologies developed in point 3 can be used in points 2 and 4. Therefore, the main focus of this work is on the reconfiguration of a robotic work-cell, which is highly conditioned by the referred hidden dependencies, namely in terms of software.

## 1.1 Problem

Complexity is present in both operations and coordination, as well as in the manual configuration of these. This is partly understandable and manageable, since a specific machine or manufacturing process may be quite advanced and complex. Complexity would then be local. However, in particular where software is involved, ad-hoc accidental dependencies between operation and coordination make the situation quite difficult. Additionally, the configuration of each machine and subsystem exhibits a variety of user interfaces and

configuration/programming tools. Large factories have highly qualified engineers that deal with this reality, but the situation in SMEs can easily become unmanageable.

We may refer to the problem as a matter of orchestration. Orchestration here means the (semi-)automated arrangement, coordination, and management of complex manufacturing systems, including their interactions in terms of communications and their services in terms of computer control. The topic can thus be formulated as finding optimal or at least feasible principles for the orchestration of operations in small-scale manufacturing.

There are fundamental issues to solve that are more than just a matter of engineering. The situation calls for a scientific approach, including careful attention to recent technology enablers being developed in other areas.

## 1.2 Approach

### 1.2.1 Scientific approach

Finding SME-suitable principles for the orchestration of manufacturing operations does not lend itself to a formal theoretical analysis, which would have required sufficiently complete formal models and derivation of (sub-) optimal solutions and their properties. It would have been preferable to have formal proof of an optimal solution, but the complexity of the equipment, the involvement of humans, the partly unstructured SME environment, the need to adhere to industrial practices, and the variety of SME application, all hinders a theoretical approach. Hence, an **empirical approach** must be taken.

The possible variability of the involved software in combination with the ever-changing production in flexible production systems poses a major difficulty concerning testing and evaluation within the necessary empirical approach. Additionally, experiments cannot (with reasonable efforts to the required extent that would give statistical evidence) be setup in actual industrial environments (where they would disturb the business-crucial manufacturing).

Therefore, experiments must be carefully selected and setup using real industrial equipment in a laboratory environment where the necessary testing can be carried out.

Still, given these conditions and the uniqueness of some equipment, it is extremely difficult to have experiments that can be repeated elsewhere, which can be a problem for the validation and acceptance of the results. However, if the suggested solutions in terms of technical principles can be found or confirmed by independently published research results, or if related techniques show up in new products, this can hopefully contribute to the validity of the results. That is, similar results for other laboratories are valuable, but the actual differences will reveal variations that deserve detailed studies.

## 1.2.2 Technical approach

Small-scale flexible manufacturing systems consist of equipment that, from a computing point of view, is distributed. The different devices are typically not engineered to work together initially, but must nevertheless be easy to configure together in the workshop. The basic approach, therefore, is to make use of software platforms that support distributed components in a flexible manner. However, existing enterprise computing platforms may not comply with the efficiency needs for embedded devices, and they may lack robustness regarding unreliable interconnections. Hence, available technologies need to be combined and utilized with some care, and suggested solutions must be compatible with and relevant to actual factory needs.

Specifically, the approach taken consists of four parts:

1. Supporting loose coupling between components to promote simple composition when equipment is installed or replaced. Interactions need to be asynchronous and event-based via well-defined and self-descriptive interfaces. These need to comprise services in terms of manufacturing operations, not in terms of internal software functions.
2. Creating unifying principles for user interaction and interfaces, thereby enabling non-expert users to configure or reconfigure and program or reprogram the production system. A user interaction that allows the combining of basic services in a compound service should result in a new service, easily accessible via both programmed and manual interfaces.



3. Model-based approaches in engineering have proven efficient for performance and reuse, but models may consist of knowledge that is too hard or too costly to obtain in a flexible and unstructured SME environment, and which may not be expressed in terms that are understandable on the factory floor. A better approach is to enable a task-based view of knowledge in terms of the manufacturing processes, with the human operator providing the intelligence via simple interfaces according to the previous item.

4. Software is by default not self-descriptive and the normal sequential execution of imperative code does not compose. Meta-level knowledge and declarative descriptions should therefore be used, if possible, without compromising the previous items. The aim is to generate the actual application-level software from such high-level descriptions.

The experimental evaluation should verify individual techniques, and results can be compared with related research. The overall approach is to combine the results from each part of the evaluation into applicable principles for future SME-suitable work-cells.

## 1.3 Organization of the thesis

This thesis is divided into 8 chapters. Chapters 1 and 2 are introductory chapters. Chapter 2 serves two purposes: one is to provide the reader with common knowledge on the most important technologies used throughout the thesis; the other is to discuss some preliminary concepts and assumptions that support the approach followed throughout the work. In section 2.1, safety-related issues are discussed: those regarding the industrial use of Ethernet devices (section 2.1.1); and the use of PC-based architectures as computing platforms (section 2.1.2). In section 2.4, service-oriented architectures are introduced and their device-level version outlined in section 2.7.

In Chapter 3, an overview of several service-oriented architectures is provided. Two architectures are selected and evaluated against a real test-bed. From this evaluation,

information is retrieved about the best features and architectural styles that should be present in a service-oriented environment to be used in an SME work-cell.

In Chapter 4, a generic solution is proposed for the integration of service-oriented robotic work-cells that is further described and implemented in Chapters 5 and 6. Chapter 5 introduces mechanisms for the automatic generation of services, with examples for voice recognition grammars (section 5.3) and industrial robot programming languages (section 5.2). In Chapter 6, a language is proposed for the orchestration of services. Programs written in this language can be seen as robotic work-cell programs.

In Chapter 7, an approach to the automatic configuration of programming-by-demonstration (PbD) robotic work-cells is taken using a descriptive language from which services and orchestrations can be extracted.

Chapter 8 introduces the work that can be done with the results from this thesis and draws conclusions.

## 1.4 Publications

Five publications refer exclusively to the work described in this thesis:

Veiga, G.; Pires, J.N.; Nilsson, K.: On the Use of Service Oriented Software Platforms for Industrial Robotic Cells. IFAC International Workshop Intelligent Manufacturing Systems (IMS'07), University of Alicante, Alicante, Spain, May 23-25, 2007.

Veiga, G, Pires, JN, Nilsson, K., "Experiments with Service Oriented Architectures for industrial robotic cells programming", Elsevier Robotics and Computer integrated Manufacturing, 2008

Veiga, G., Pires, J.N., PLUG-AND-PRODUCE TECHNOLOGIES, On the use of Statecharts for the orchestration of service oriented industrial robotic cells, ICINCO International Conference on Informatics in Control, Automation and Robotics, Madeira, Portugal, May, 2008.

Veiga, G., Pires, J.N., "USING ROBOT PROGRAMING LANGUAGES TO SPECIFY SERVICE CONTRACTS", Proceedings of Controlo 2008, Special Session on Robotics Applications, Vila-Real, Portugal, 2008.

Veiga, G., Pires, J.N., Nilsson K. , "Automatic Configuration for Programming by Demonstration robotic Work cells", (to be submitted).

One publication refers to work partially related with this thesis:

Pires, JN, Veiga, G, Araújo, R, "Programming-by-demonstration in the coworker scenario for SMEs", Emerald Industrial Robot, 2008

One publication refers to work in related fields:

Veiga, G. and Cancela, R., "Advanced HMI for robot programming: An industrial application for the ceramic industry.," *Proc. International Symposium on Robotics.*, Barcelona: 2009.



*I have to respect the strictly limited size of my head  
and can deal with only one thing at a time*

*Edsger Dijkstra*

# 2 Background

---

## 2.1 Predictability and safety

### 2.1.1 Ethernet networking and real time

The Ethernet use in industrial automation has been a research subject in academia for more than 20 years, with special focus on its real-time capabilities/limitations.

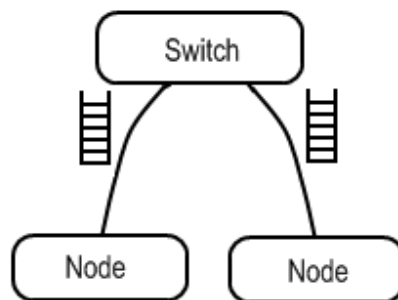
These efforts were the subject of a remarkable revision by Decotignie in 2005 [4], which detailed all technical aspects of the most significant solutions. Decotignie defined an interesting metrics to evaluate Ethernet-based real-time solutions with respect to their tolerance to non-real-time traffic (Table 2).

**Table 2 Classification of Solutions (Adapted from [4])**

Class	Behaviour in presence of 802.3 compliant nodes	Examples
Non interoperable	Don't operate properly	FTT-Ethernet
Interoperable homogeneous	Operate properly but loose temporal guarantees	RETHET, traffic smoothing
Interoperable heterogeneous	Operate properly and keep	EtherReal, switches + priorities

This classification provides a real benefit measurement of the Ethernet use in industrial automation, since many of the so-called *Industrial Ethernet* solutions are dependent on special hardware and don't tolerate regular traffic, which makes them proprietary solutions with standard connectors. Between the *Interoperable heterogeneous* approaches, the use of full-duplex switched Ethernet is emerging as the most promising approach [5][6][7].

Full switched Ethernet is standardized by the IEEE 802.3x standard [8], which basically **removes the possibility of collisions** in the Ethernet communications. Previous versions of the Ethernet were intrinsically non-suitable for real-time communications since they relied on a *medium access control* (MAC) algorithm that used collision detection to manage the access to the communication medium (CSMA/CD - *Carrier Sense Multiple Access with Collision Detection*). On the other hand, the so-called *full-duplex* Ethernet [8] uses a topology where every single node is connected to an Ethernet switch that isolates every node collision domain. An Ethernet switch works as a MAC bridge, that is, the switch regenerates the information and only forwards it to its specific destination port. It is important to notice that a frame **can still be delayed inside a switch** if there is another message being sent through the same port.



**Figure 1 Switched Ethernet and communication buffers**

According to the IEEE802.1D specification [8], the switch (or a station) can send PAUSE frames to notify the other side of the link to suspend the message sending, thus preventing the overflow inside the switch (or inside the NIC-Network Interface Controller station), but this

delays communications. Other emerging standards around Ethernet technologies with relevance to real-time performance are the IEEE802.1Q [9] and IEEE802.1P [10]. The first one defines the creation of VLANs (*Virtual Local Area Networks*) within the same switch, which allows traffic isolation between logically defined networks, and the second allows traffic prioritization up to 8 different traffic classes. These enhancements over Ethernet technology allow a predictable communications timing and have sustained several research projects that achieved many promising results, using either stochastic approaches [11], scheduling algorithms [12][13], network calculus [14][15] or feasibility analysis [16]. Even though a number of these proposals need some kind of extra (like a switch firmware upgrade or similar), it can be pointed out that **the industrial use of standard Ethernet as a real-time communications framework is close.**

It is important to stress that the real time performance of switched real-time Ethernet has limits that are inherent to topology, the size of Ethernet headers and the lack of precision timing. Precision clocks are now ruled by a common standard (IEEE1588) but due to their low interest to office applications, only few devices support it.

These limitations have led the market of field-bus makers to propose *Ethernet-based* real-time solutions. In a market movement that mimics the old field-bus battle, companies are fighting and claiming that their solution is *the Industrial Ethernet*, and standardization entities have overcome the diversity of platforms with a profile solution for the IEC 61784 [17] that glues all of them together.

**Table 3 Real-time Ethernet profiles defined in IEC 61784**

IEC 61784 Profile	Brand Names
CPF-2	ControlNet (Ethernet/IP)
CPF-3	PROFIBUS/PROFINET
CPF-4	P-Net
CPF-10	Vnet/IP
CPF-11	TCNet

CPF-12	EtherCAT
CPF-13	Ethernet POWERLINK
CPF-14	EPA
CPF-15	MODBUS-RTPS
CPF-16	SERCOS

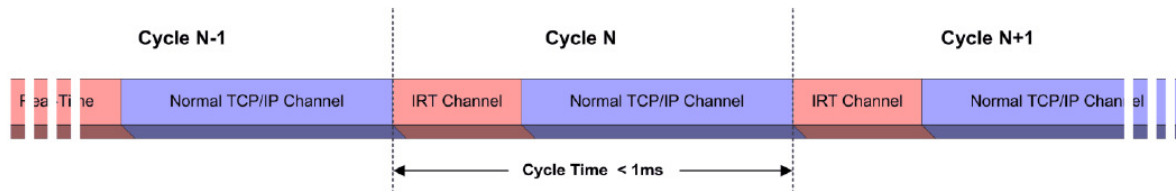
Comprehensive overviews of these Ethernet profiles are made in [18] [19] [20], proposing different classification structures on their delivery times. From the author's point of view, only two different groups should be delimited:

The **first group** ranges from 10ms to 100ms in delivery time, and includes all technologies that exploit research results presented before for the **full-duplex switched Ethernet technology**. The most relevant ones (that have to this day become products) are: Ethernet/IP [21], PROFINET-RT [22] (one of the profiles of the PROFINET solution) and MODBUS-RTPS [23]. These technologies implement real-time specifications over switched Ethernet and use different strategies to address Ethernet limitations. To obtain timing precision, for example, both Ethernet/IP and PROFINET-RT rely on IEEE1588-based protocols [24], while MODBUS-RTPS uses an UDP-based heartbeat mechanism. To deal with priorities, these protocols use the priority scheme at the Ethernet MAC layer, according to the IEEE802.1Q specification [10]. This group targets all applications from manufacturing systems to distributed IO, but it is normally excluded for motion control applications.

The **second group** targets motion control or safety systems, and the most significant examples are EtherCAT[25], the second specification of PROFINET, PROFINET IRT [22], PowerLink [26], and Ethernet/IP CIP with CIPSync [21]. The first two specifications modify the lower layers, passing through TCP/IP (*Transmission Control Protocol/Internet Protocol*) layers with their own scheduling schemes, and reach cycles below 100µs to control more than 100 coordinated axes. The third, Ethernet/IP with CIP Sync, is closer to the standard Ethernet but requires the implementation of the IEEE1588 protocol in specific hardware to meet cycle times above 1ms. These solutions are dependent on specific hardware but are somehow interoperable with

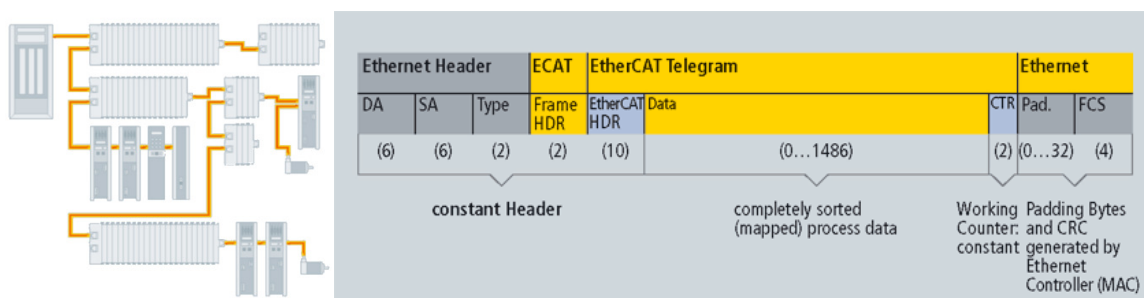


normal TCP/IP or UDP traffic. Figure 2 shows how the isochronous<sup>1</sup> real-time version of the PROFIBUS profile divides the available time in a real-time channel, leaving the rest of it free for non real-time traffic. The real-time channel width can be defined by the systems engineer. PowerLink [26] uses a similar time-slicing mechanism.



**Figure 2. PROFIBUS-IRT channel division [22]**

*EtherCAT* uses a Master-Slave architecture with passive slaves that only process and forward information in special-ring topology (Figure 3). Slave Ethernet frames with a specific *Ethertype* (0x88A4) and every station in the net removes and adds information into the *EtherCAT Telegram* (Figure 3). Any outside traffic is transparent to the *EtherCAT* network. An *EtherCAT* message can also be transported directly in the data section of an UDP datagram, via IP, which enables the use of *EtherCAT* through several subnets but compromises hard real-time performance.



**Figure 3. EtherCAT's topology and channel division [48]**

<sup>1</sup> The isochronous term refers to a communication with a guaranteed throughput and is used by the PROFIBUS foundation to refer their below millisecond profile.

Within this scenario, it is predicted that some of these technologies continue being used in parallel in a **two Ethernet network level structure**, one level that includes **hard real-time elements** when they are needed, like safety and motion control, and another that manages the **rest of the plant communications**, including higher layers like p. e. management. This situation is in fact an evolution from the actual hybrid network present in most industrial applications nowadays, which rely on two different networks to deal with work-cell communications: one is based on a traditional field-bus platform, and provides real-time support, integrates safety devices and is usually connected directly to the robot controller or to the PLC present in the work-cell; the other is based on Ethernet and allows the integration of advanced PC-based interfaces, integrates several non real-time devices and is usually connected with the office level for error logging and history reporting. Another important factor in favour of the adoption of two levels of network communications is the rate of applications that really need below millisecond real-time performance. Considering a rough estimation made in [27], only 10% of total applications will call for motion-control, therefore leaving much space for the growth of other technologies (non-Isochronous), which are more competitive in terms of cost due to higher compatibility with office components, and which will thus easily become *de facto* standards.

Considering **industrial robotics** applications, one can classify as uncommon the need for real time in the communication framework. In some special but important cases, such as visual servoing and conveyor-tracking involving feedback loops via several interconnected devices, real-time systems need to be able to accomplish shop-floor deterministic traffic, but it would be uncommon that those cases should require Isochronous real-time behaviour.

All aspects described above indicate a **strong presence of standard Ethernet** in future plants, which supports the option for Ethernet networking throughout this work.

### 2.1.2 PC-based control, programming languages

Safety is becoming increasingly important for robotic work-cells, given the intense utilization of safety sensors and the trend of removing the fences around the machines. This might seem

contradictory to the use of PC-based software for cell control, but two facts simplify the scenario: safety sensors and controllers are configured separately, based on special hardware and certification procedures; safe robot work-cells are not mission critical (like an airplane control system is). So occasional failure can be acceptable if it can be detected and handled (by stopping or performing another task). Therefore, we can simplify the (hopefully fast) development of flexible manufacturing systems by avoiding some of the issues of X-by-wire and similar systems for vehicle technologies.

## 2.2 Robotic work-cell

The groups of technologies that complement a robot compose the industrial robot cell. An industrial robotic cell is a networked environment where several devices contribute to perform a given task. To accomplish the idea of a robot co-worker adapted to the SME environment, the definition of the SME industrial robotic cell must be quite different from the traditional one. The traditional robotic work-cell is normally integrated in a *Flexible Manufacturing System* (FMS) and consists of a functionality unit that is part of a wider manufacturing system. This unit works in a nearly automatic way, targeting the best possible time cycle, relying on an advanced communications system to receive manufacturing orders, reporting alarms, producing log files. In their traditional usage, robots are usually programmed with the production line stopped and with a great amount of specialists around, from programming to mechanics. On the other hand, a robotic work-cell as an SME mechatronic co-worker should be “only” semi-autonomous but highly reconfigurable and reprogrammable. Bearing this in mind, some important features of modern robots, like for example, good integration in *Supervisory Control and Data Acquisition* (SCADA) systems or time cycle can be considered less important than (re)programming or reconfiguration time. From an SME point of view, here are some features of the future robotic cell:

- Better human interaction.
- Extensively reconfigurable.

- Easier to program (both in the device level and in the robotic cell level).

### 2.2.1 Human/robot interaction

Human-machine interaction is a widely discussed subject with crucial importance in the success of machines. In the past, human-machine interaction was explicit in the machine, like in a saw or in a conventional lathe, for example, it is explicit where the hand should grab and what is supposed to be done. With the arising of computers, things got worse because systems became more complex and, in the majority of cases, a *Graphical Human Machine Interface* (GHMI) is now the only way to provide information. The success of a product, especially considering software products, is related to the efficiency of the human-machine interaction. A good example of this fact is the *Personal Computer* (PC)'s usual configuration, with mouse and windows-based graphical interfaces, which were born over 30 years ago and influenced all software developed afterwards. Industrial robots are one of the few types of industrial equipment with explicit functionality, due to the resemblance of the robot with the human arm. Unfortunately, this physical resemblance has not been extended to the interaction level. Regarding the importance of the human-machine interaction in industrial automation, robotics is a particular case, due to the fact that robots are mainly programmed online. One of the main reasons for this difference is the relative lack of success of the offline programming tools. One way to evaluate the success of robot offline programming tools is making a comparison with the Computer Aided Manufacturing (CAM) technologies, which in most cases generate a Computer Numerical Control (CNC) code that is used in machines with similar functionality. The lack of volumetric accuracy and the absence of a strong standard in terms of intermediate language (in analogy to the G-code) can be considered possible causes for this relative failure. Although offline programming tools have their important (growing) market, industrial robots are still very dependent on online teach pendant programming. This technology has suffered a great evolution over the past years [28] but the basic concept is still the same, and it is not well adapted to the SME scenario described above. Therefore, new paradigms for online robot programming are needed, with special focus on **explicit programming techniques** that can bring the worker closer to the robot. Recent developments presented by a robot manufacturer

have embedded high speed force control feedback into the robot controller [29], which provides good perspectives in terms of advanced sensor integration, crucial to improve the man/machine interaction. These systems have been around in research for a long time [30] but their industrialization opens possibilities in mass products with advanced teaching techniques, namely PbD [31] and path learning [32].

### 2.2.2 Reconfiguration of a robotic work-cell

The **reconfiguration of an industrial robotic cell** should be easily accessible to the SME worker, without requiring the recurrent presence of the system integrator. These devices should interact with the robot controller in a nearly automatic way and provide the user with a friendly experience.

The term *Plug-n-Produce*, coined in 2006 [33]<sup>1</sup>, draws its analogy from the *plug-n-play* experienced by users in the office environment. The Plug-n-Produce concept is made up of 3 layers: *application, configuration, and communication*.

*Communication plug-n-produce* is the lowest level of the systems and deals with communications among the robotic work-cell, which includes protocol standardization, messaging methods and networking architectures and features, like p.e. discovery and eventing.

*Configuration Plug-n-Produce* is the intermediate layer and manages the settings that the users should not need to care about, like bandwidth requirements or default values. Furthermore, this layer should also deal with cell specific information, like the I/O mappings and basic maintenance operations, among others.

*Application Plug-n-Produce* delivers the user high-level functionality corresponding to the robot cell capabilities. The presence/absence of these devices in the industrial robotic cell should be reflected in the functionality provided by it. In this matter, the use of semantic knowledge referring to the industrial robotic world has been the target of extensive research leading to

---

<sup>1</sup> The term plug-n-produce is also used in robotics for assembly operations, referring to problems associated with physical reconfiguration[182]

several European projects over the last few years [34][35]. These projects are making full use of the momentum around the semantic web technologies, which provide well-established languages and development platforms like the *Ontology Web Language* (OWL) [36] and the protégé-owl development platform [37].

The work developed for this thesis can be framed in the layered structure presented above in the following way: chapter 3 revises existing network architectures that can be used as *communication layer*. Chapters 4, 5 and 6, provide two alternative/complementary approaches to the *application layer*. Finally, chapter 7 presents an approach to the challenges raised by the *configuration layer*.

### 2.2.3 Industrial robotic work-cell programming

An industrial robotic cell can be viewed as a distributed environment with multiple dedicated processing units. The reconfiguration of these systems is strongly connected to the existence of industrial robotic cell programs that can act as glue between dedicated systems. The idea of industrial robotic cell programs has been present since the introduction of the object-oriented approaches for industrial robotic cells [38]. This approach has been successfully followed by several authors [39], evolving together with component-based solutions, leading most robot manufacturers to provide similar packages [40][41]. Even nowadays, research on robotic cell programming is made using object-oriented technologies [42][43]. However, this approach has several drawbacks: **knowledge of traditional programming languages** (C++, C#, Java) is needed, which can be hard to find among robot programmers; due to the computational capabilities of this type of languages and to the characteristics of component-oriented programming it is usually difficult to promote the separation of concerns, computation and coordination, leading to coordination programs that can hardly be reused. In this thesis, these issues are addressed through the use of service-oriented architectures, together with the definition of a robotic work-cell programming language.

## 2.2.4 Holonic cell structure

A **holonic cell structure** [44] is considered in this work, with *holons* composed by automation devices or components with some level of computing power, like an industrial robot or a vision system. These holons can be hardware devices, but also software components that provide functionality with a specific interface. Previous experiences from the author and his research laboratory [45][46][47][48] have shown that the requirements of advanced human-machine interaction, like the ones present in SMEs, ask for many office devices, like speech recognition systems, digital pens [48] and digital tablets [45], to be integrated into robotic work-cells. This integration usually requires ad-hoc software platforms, generally based on PC hardware. In this approach, the distributed environment needs to integrate these devices and their respective software as *holons*.

## 2.3 Middleware

A middleware serves to promote the task of designing, programming and managing distributed applications by providing a simple, consistent and integrated distributed programming environment. A middleware is a distributed platform which abstracts over the complexity and heterogeneity of the underlying distributed environment with its multitude of network technologies, machine architectures, operating systems and programming languages. In the following sections (2.4, 2.5), two of the most important characteristics of a middleware are analysed: the **functionality abstractions through the use of service oriented architectures** and **messaging patterns**.

## 2.4 Service oriented architectures

Looking briefly into the history of software engineering<sup>1</sup>, we can see that in the 60's the first high level languages introduced the notion of compiler. The **compiler** represented the **decoupling of the code developed and the target machine**, and materialized a first degree of abstraction. The problem with this type of languages, like COBOL, for instance, is the lack of a structured programming model [49], which makes their programs dependant on their own structure. In the 70's, languages like *Pascal* or *C* **decoupled the program from its structure**, once again raising the level of abstraction with the introduction of **functions and data structures**. With the increasing size of programs, problems regarding code maintenance and reusability led developers to the introduction of object orientation, with languages like C++ or Smalltalk implementing concepts stated years before [50]. **Object Orientation** provided the user with a way to package functions and data inside an *object*. Objects are constituted by methods (former functions) and data, and are typified by classes, which **are the basic unit of reusability**, directly or through specialization, and enable **domain modelling** through the form of a class hierarchy. Object-oriented programming also had its limitations, namely: the class as the basic block of reusability is tied with language, which means that portability is bound to the language used; implementation is not a clear process, since a C++ program, for example, does not specify what the binary data looks like.

Object-orientation issues have been addressed over time with different technologies that have defined important industry standards, like *Microsoft's static and dynamic link libraries (.lib and .dll)*, which bridge code to binaries.

The next big stage in software evolution can be considered component orientation, where the portability of software has achieved a new level. The World Wide Web Consortium (W3C) states that a *component is a software object, meant to interact with other components, encapsulating certain functionality or a set of functionalities. A component has a clearly defined interface and conforms to a prescribed behaviour common to all components in the*

---

<sup>1</sup> This historical approach does not intend to be exhaustive and skips some important advances in computer history, but it provides a generic view of a programmer over the last thirty years.



*architecture*. One important paradigm shift implied in this architecture is that building software systems relying on the composition of standard components enhances reusability. Component orientation defines a set of interfaces that allow programming against an abstraction of the service: the contract. Another important feature is that contracts rely on a binary type system, which allows components to be developed in different languages. Microsoft's COM (*Component Object Model*) is a component-oriented platform introduced in 1994 and is a successful example of this feature: components can be developed in C++ and then used by less skilled Microsoft's Visual Basic programmers.

*Microsoft's COM and CORBA* were among the most significant component-oriented platforms in the early days, but both had a limited success due to implementation details.

CORBA was in the late nineties an advanced software middleware driven by a consortium with many interesting features, like cross-language and multiple platform support. Among several non-technical reasons for CORBA's lack of success, there were some problems in the quality and the time-to-market of the standards produced. The late arrival of real standards like the ORB *Corba Component Model* (CCM) compromised portability and interoperability [51], and when they finally arrived, they were so complex they led vendors to bad implementations.

Microsoft's *COM* addressed some important issues from old dll technologies, like p. e. versioning, but COM components were hard to program. One of the reasons for this was the inadequacy of the languages used to write components (C++ and Microsoft Visual Basic), since they are object-oriented and not component-oriented. This brought about the need to use bridging platforms like ATL (Active Template Library), which introduced an unnecessary ugly model.

The internet era changed the world of software development. While Microsoft extended COM's model to create OCX (*OLE Custom Controls*) and *ActiveX* components, the appearance of *Sun's Java Programming Language* revealed a new path in software development. **Java** is friendly and powerful, and the intermediate language **Byte-code defines a sandbox** of functionality that limits the access to low-level resources and is therefore inherently safe. This fact was one of its major wins in the internet race, since ActiveX/COM/OLE's natural lack of safety made it less

adequate for browser use. But the Java technology has brought another important aspect to software engineering: **safe programming**, a set of language features that limits the number of programming bugs that are explicitly caught neither in the compiling nor in the runtime stages. After the “internet war”, both Microsoft and Sun moved their resources to the **definite solutions on component-oriented platforms**. Sun Microsystems developed Enterprise Java Beans (EJB), and Microsoft presented the .NET framework, which, unlike COM, provides: safe programming (both from the programming and the access to resources points of view), and defines standards for many issues like metadata sharing, serialization and versioning.

These last technologies have faced mass use on large distributed business systems, and programmers have come across several problems. The first and the most significant of these concern **multiple dependencies**:

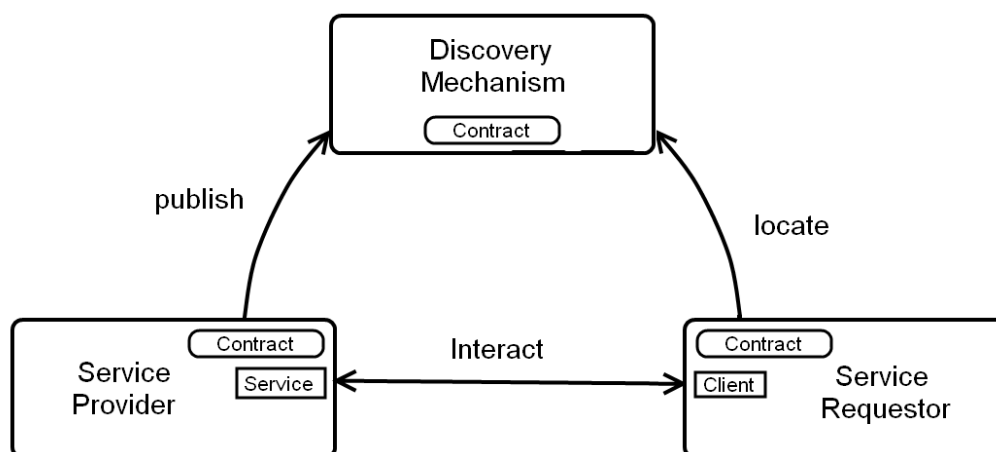
- *Operating System dependencies* - for instance, despite notable efforts [52], .NET is still mainly a *Windows* platform.
- *Intermediate platform dependencies* – Java Beans or .Net framework components are not interoperable.
- *Language dependencies* – If you want to develop a Java Bean component you are limited to Java.

Besides these dependencies, there are other issues, like the **standardization of transaction management, communication protocols, and security**, among others. Some of these problems have already been addressed by a few previous component-oriented platforms: transactions, for example, were addressed by *Java 2 Enterprise Edition (J2EE)* with *Java Transaction API (JTA)*, COM+ with *Microsoft Transaction Server (MTS)*. Like before, however, these issues represented a global problem that needed a global approach, free from inherited problems. That is the moment for the appearance of *Service-Oriented* programming.

With the advent of world scale Internet, **service-oriented architectures (SOA)** emerged to increase the degree of **decoupling between software elements**. Decoupling has been the major player in computer engineering, from languages to platforms. Each time software evolves, it

reaches a new level of decoupling. It is important to notice that coupling is necessary, since it represents functionality, but as soon as systems grow, decoupling is needed to reach higher levels of reusability.

An SOA relies on highly autonomous but interoperable systems. Three major players compose the service-oriented architecture pattern graphically presented in Figure 4: the service is offered by the **service provider**, which publishes its functionality in a discovery agency in the form of a **service contract**. The **service requestor** searches for a given service through the discovery mechanism and, when an interesting functionality is found, retrieves the service provider's location from the service contract and starts interacting directly with the service provider.



**Figure 4. Service Oriented Architecture**

The architecture pattern presented in figure 4 configures a general architecture, and many different variations can be extracted from this basic topology, when implemented in the specific middleware.

A key element in this architecture is the **service contract**, which supports not only the uniqueness of a given service but also the description of the interface available for the client. The client uses this information to generate software that matches the respective service.

The **discovery mechanism** can be implemented via a **contract repository**, where contracts are stored by providers and searched by requestors, or via **peer-to-peer** messages for the location and publication of services.

There are also several different mechanisms available for the **interaction** between the service requestor and the service provider. The two biggest families of *Web Services*, *WS\** and *Representational State Transfer* (REST), define different interaction mechanisms, that are further analysed in sections 2.4.1 and 2.4.2.

The **definition of a service is ruled by the larger context**; this means that all technological details are hidden, but also that the concept that supports the service is more business (or process)-related and less technological-related. An *SOA*-based middleware should provide software engineers with time to focus more on the business logic and less on the plumbing details. One important advantage of the *SOA* technologies is that the developments have been driven by concrete needs raised by several implementations of component-oriented distributed platforms. Therefore, an increasing number of companies has been trying to avoid ad-hoc plumbing and relying on standards, which has increased the importance of standardization organizations, like the W3C consortium [53] and the OASIS [54] (*Organization for the Advancement of Structured Information Standards*) forum.

## 2.4.1 WS\* - Web Services

The technology of *WS\*-Web Services*<sup>1</sup> is the most used connection technology of service-oriented architectures with wide implementation on the Internet. *WS\** services essentially use XML to create robust document-based connections, and are the most visible face of a wide *SOA*. Web Services are in their first adoption phase, but there are some facts that predict a wider adoption: they are sustained by a consortium, are by nature platform-independent, and almost all major industry players are part of the process (either in the consortium or with their implementations). A good example of this is the *Web Services Interoperability Technologies* (WSIT, previously known as *Project Tango*), that joins together two of the major players of

---

<sup>1</sup> *WS\** Services are commonly called just Web Services, but in this text this expression is avoided to clearly separate them from REST Web Services.

modern computing, Sun Microsystems and Microsoft, in an effort to test current *Web Services* implementations to access their interoperability.

The mechanism for service discovery and location specified by WS\* is defined by the *Universal Description and Discovery Integration* (UDDI) standard [55]. The UDDI defines a repository (web location) of service contracts, where a client can search for available services and retrieve the respective interface definition.

The *Web Service Description Language* (WSDL) describes the WS\* service interfaces by defining the structure and sequencing of XML input and output messages [56]. The invocation of services is ruled by the SOAP<sup>1</sup> specification [57].

WS\* specifications are composed by several documents that regulate different parts of service communications, from messaging basics with SOAP to security with *WS-Security*. For a complete reference, please see [53].

The **interaction patterns** between a service consumer and a service can be grouped into four major types: the **one-way message**, when the sender does not care about the message and the operation doesn't have a return value; the **request-response** and the **solicit-response** patterns, that form pairs of RPC (remote procedural calls) either directly (solicit-response) or inversed; and the **notifications**, that allow the client to be informed when something happens on the service side.

The request-response and the solicit-response patterns are synchronous patterns, while the one-way message and the notification patterns are inherently asynchronous. Different types of patterns can be present in the same WSDL contract. It is important to notice that the asynchronous behaviour of such patterns is affected by the underlying layers, such as the transport layer. An http request, p.e., always requires a response and every asynchronous message pattern developed in the service layer will therefore behave synchronously.

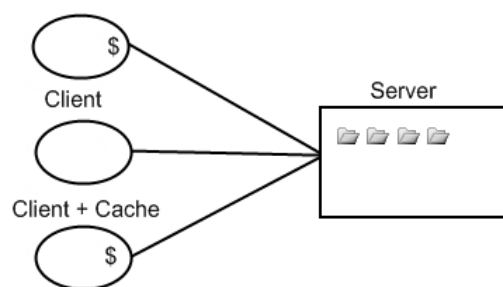
---

<sup>1</sup> The SOAP 1.2 specification dropped the acronym for SOAP – Simple Object Access Protocol, due to its misleading nature. This text follows these guidelines.

From a programmer's point of view, the design of a service can be taken on in two principal ways: by using an object-oriented language and deriving operations from class methods, or by composing an *XML* document directly. The first type, often called *RPC* web services, is closer to the distributed component-oriented programming and is better accepted due to its simplicity. It is important to stress that, although the name *RPC* can be associated with some type of synchronicity, this is just a tool for service design, and the underlying message patterns can be either synchronous or asynchronous. Furthermore, the key element of the *WS\** web services definition is the contract, which is completely detached from any code implementation. As Vogles points out in [58], a *WS\** service is a software element that can process XML documents, independently of the way this software is implemented, either through object-oriented techniques or not.

## 2.4.2 REST Web Services

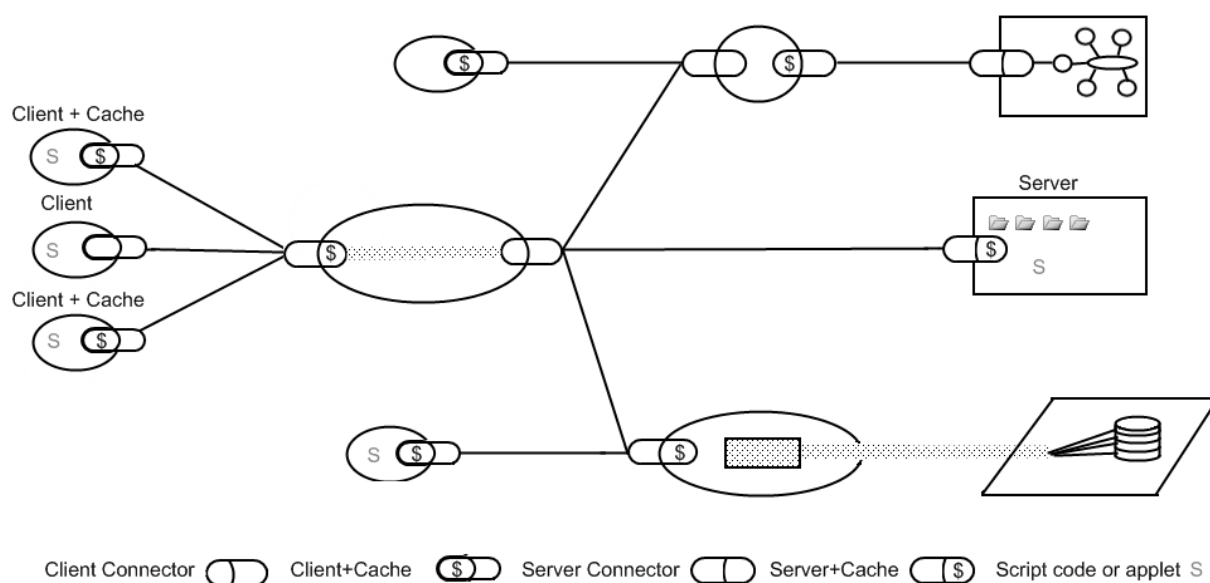
REST is a style of software architecture engineered for distributed systems. This architectural style was defined by Roy Fielding [59][60], who studied the fundamentals that supported the great scalability verified in the Web infrastructure, and established a set of implementation principles from them. REST is defined in [59] as a **client server, stateless, cacheable protocol**.



**Figure 5. Client-server, stateless, cacheable protocol (adapted from [59])**

Considering the web technology, this type of protocol means that each HTTP message includes all the information needed to complete the request and, as a result, neither the client nor the server needs to store state information. This fact boosts caching and intermediate messaging capabilities, since all messages can be treated without any previous knowledge.

REST also specifies that both the application's state and functionality are abstracted into **resources**, which are **uniquely addressable** with URIs (Universal Resource Identifiers). The access to those resources is constrained by a set of well-defined operations (*POST*, *GET*, *PUT* and *DELETE*), in order to define a **uniform interface**. On top of this, Roy Fielding defined that the protocol should be **layered**, in order to achieve a good performance in Internet-scale requirements, and finally added the support for **code-on-demand**, which enables the client to download executable code in the form of applets or scripts. The result of this composition is presented in Figure 6.



**Figure 6. REST architecture (adapted from [44])**

It is important to notice that, despite having supported the easy growth of the web network, some of these principles have not been commonly followed. For example, the use of cookies to establish stateful sessions is very common and violates the concentration of the state information in the URI.

REST intends to support the design of applications that should behave as a network of websites (a virtual state), where the user can interact with the application by selecting transitions, and getting as a result the next page that represents the new application state. A representation consists of data and metadata describing the data. As described in a dissertation by Roy

Fielding, REST is an "architectural style" that basically exploits the existing technology and protocols of the Web, which, when applied correctly, should provide a scalable development platform.

The REST style has been used in the implementation of web services and, although its adoption is not comparable with WS\* technologies, there is a verbose community that claims its benefits against WS\* architectural styles. This discussion is known as "RPC vs REST", and is biased by the fact that REST supporters consider that all WS\* implementations are strictly made through an RPC blocking model.

The uniform interface required to create a REST application defines an important difference from RPC, one of the most common messaging methods to reach a server, like in WS\* technologies p. e. A REST application limits the number of verbs (operations) to POST, GET, PUT, relying in the variety of nouns (messages) to achieve functionality.

From a web programmer's point of view, REST defines a set of rules to follow while implementing web applications. These rules specify how web standards, such as HTTP methods or URIs, should be implemented in order to allow scalability and consistency. This consistency should be enough to support the transfer of these technologies from machine-to-human interaction, like the one in current web technologies, to the machine-machine technologies that are the focus of *REST Web Services*.

## 2.5 Reactive systems

Traditionally, a computer program is like a black box, with an ordered sequence of input processing, output and termination steps. These systems are called *transformational systems*, because the relation of the input to the output is sufficient to completely characterise the behaviour of the program. On the other hand, reactive systems [61][62][63][64] are those which cannot be completely characterised in terms of the relation between input and output. Transformational programs are essentially data-driven, in that it is the flow of data which



controls the system's behaviour, by contrast with reactive systems, which are control or event-driven. Consequently, reactive systems are interactive by nature. They will typically receive some initial input, but then continue to interact with their environment during the course of their execution, both by sending values (output) to and receiving new information (input) from the environment. Reactive systems also include those which are real-time, embedded, concurrent and distributed.

## 2.6 Messaging patterns

In this section, some of the most relevant middleware messaging patterns will be revised.

An important definition when specifying messaging patterns is the definition of synchronous. The definition of synchronism concerns the program flow. A procedure or system call is considered synchronous whenever it blocks the caller until the response is ready, and asynchronous when the caller is not blocked and can later retrieve the response. This definition extends the notion of "method invocation" to a distributed context and is tenuous, especially considering modern operating systems, where threads are easily accessible and, when combined with synchronous calls, provide asynchronous behaviour. Some authors [65] prefer to define synchrony through the definition of *transaction*. The transactions paradigm defines a commit-or-abort behaviour, thus providing guarantees of consistency both for the service provider and for the service consumer.

### 2.6.1 Remote Procedural Calling RPC

*Remote procedural calling* (RPC) is one of the best-known forms of distributed interaction, and relies on the notion of method invocation extended to the distributed context. This type of interaction was first proposed in the form of remote procedural calling for procedural languages, and later exported to the object-oriented programming environments. Many of the currently working distributed applications, such as *Java RMI*, *CORBA* and *Microsoft DCOM*, have been developed using this type of platforms.

Programming synchronously has always been known for its simplicity since, for the programmer, it was straightforward to program distributed applications by transparently applying programming structures from object-oriented programming and even procedural programming. Whereas asynchronous systems deal with complex state managements, synchronous systems can rely on transparent communications and construct logical applications with a simple program flow. However, this simplicity has a cost, related to the performance, reliability and scalability of the system. RPC introduces a very high degree of coupling, both in terms of time and flow (from the consumer's side).

## 2.6.2 Asynchronous RPC evolutions

Several attempts have been made to reduce the coupling of such systems. *One-Way* messages have been defined in all major platforms and are essentially fire-and-forget messages. In order to obtain some asynchronous behaviour, remote procedural calls have also been implemented, with the use of two different *one-way* messages, the first sent by the client to the server with the information about the request, and the second with the request's result. Furthermore, the referred access to threading in modern operating systems and the safe constructs of modern languages, try-catch and the like, provide numerous ways to achieve safe asynchronous behaviour. Both these strategies are used in every major distributed system, and in every SOA platform.

## 2.6.3 Publisher/subscriber

The **publisher/subscriber pattern** represents a family of patterns that are also called **event-driven** or **notification-based** patterns. In a publish/subscribe interaction the subscribers (consumers) have the ability to specify their interest in an event or pattern of events, and as a consequence of this subscription, the consumer is later notified of any event produced by the publisher (producer) that matches this subscription. This model can rely on an external service that provides the system with management of subscriptions, but also with storage and routing of notifications. In other cases, the communications can be realized in a peer-to-peer manner, in which publishers are responsible for the brokers' activities.

The **observer pattern** belongs to the publish/subscribe family in which subscribers register their interest directly with publishers, which manage subscriptions and route notifications. This pattern is very simple and is an extension to distributed systems of an extensively used model in programming graphical user interfaces, like in *Java SWT* or with *C#* events, due to its ability to deal with event-driven systems.

Important variants of the publisher/subscriber pattern can be classified in terms of the filtering used to specify the subscriptions. In internet-scale environments, the type of messages to be received by a given subscriber can be specified using complex options [66]: **topic-based**, that relies on the notion of subjects that can be represented by keywords, for example; **content-based**, permitting the subscriber to receive messages that match a subscription pattern that includes information from internal properties of the service; **type-based** filtering is based on the type (class) of the object sent as a message. Another important classification of the event-driven systems can be made by taking into account the **degree of decoupling** between consumer and producer [66] [67]. This decoupling can be measured in terms of: flow decoupling, referring to the synchrony between the request and the delivery of the information; time decoupling, which refers to interactions in which the parties don't need to be participating at the same time; space decoupling, referring to interactions where publishers don't have any concrete reference to the subscribers involved. It is worth noticing that these levels of decoupling target highly heterogeneous systems, internet scale, p.e, and have several implementation drawbacks [67].

There are many examples of publish/subscribe platforms in numerous domains, and a growing interest in their use for internet-scale systems is arising. *Message-oriented middleware* (MOM) and the *WS-Notification* standard are notable examples.

## 2.7 Device-level SOA

Device level SOA are specialized SOA platforms for **resource constrained devices**, normally targeting specialized market areas. These platforms usually have specific implementation

details and some extra functionality over the basic structure of internet scale service-oriented platforms, namely: **peer-to-peer discovery mechanisms; Publish/Subscribe messaging patterns.**

In the **office environment**, device level service-oriented architectures are pointed out as the support framework to extend plug-n-play functionality provided by *Universal Serial Bus* (USB) devices to the Ethernet. Due to the fast fall of Ethernet connector and controller prices, printers, scanners, projectors and all the embedded office equipment will become networked in a very short time. The discovery of these devices and their configuration/use in a standardized way is a challenge.

In the **home environment**, the rising number of automation equipments renders market standardization chaotic, therefore demanding a rich and interoperable architecture.

To address both office and home challenges, some SOA for the device level platforms were proposed around the year 2000: *Universal Plug-n-Play* is a large consortium proposal [68] that targets mainly the home environment; *Jini* is a Java-dependent middleware [69] that was initially developed by Sun Microsystems and later transferred to the open community Apache River; *Open Services Gateway Initiative* (OSGI) [70] is an alliance-based proposal, founded by companies like Ericsson IBM. Although these technologies have gained some traction, their adoption has not been massive, and the main reasons for this were:

- Market dynamics – home technologies are evolving but things take time;
- Consortium size – the lack of strength of the consortium behind the middleware made OSGI lose strength;
- Occasional facts – UPnP had its image widely damaged due to a non-safety implementation of the protocol under *Microsoft's Windows XP*;

With the big momentum around *WS\** web services in the business level, and the problems with the acceptance of *UPnP*, a new proposal has been made by the same consortium to W3C: *Device Profile for Web Services* (DPWS) is a specification with close relations with UPnP, but which uses some standards that are in line with the *WS\**. This technology is gaining great

momentum in the home electronics market, but also in research institutes and universities. Since 2003, there have been several European research projects related with service-oriented technologies for the device level. The SIRENA project [71] (*Service Infrastructure for Real time Embedded Networked Applications*) has exploited service-oriented frameworks for embedded computer environments, including: industrial automation, automotive electronics, home automation, and telecommunication systems. This successful project has opened a large number of possibilities and derived into several spin-off projects, namely SOCRADES [72], focused on extending SIRENA with semantic knowledge, and SODA [73], providing tools for dissemination and engineering.

Besides DPWS, closely related with the WS\* technologies and architecture styles, it is important to refer different service architectures.

One of the most recent service-oriented middleware has brought the REST style to the device level: *Decentralized Software Services Protocol* (DSSP) [74] is part of Microsoft's initiative on robotics, *Microsoft Robotics Studio* (MSRS). This initiative bundles inside a single package the referred service library, a concurrency/coordination programming environment with respective runtime and a physics engine, providing a complete framework for work on robotics targeting all markets: academia, hobbies and even industrial automation.

### 2.7.1 Industrial use of device-level SOA

The industrial automation engineers' need for effective distributed applications has fuelled a race for new technologies for many years. As stated in the previous section, device level service-oriented platforms usually define a set of complementary technologies to the fundamental service-oriented tenets. The combination of service-oriented styles with Publish/subscribe messaging patterns constitutes an interesting approach in defining a middleware to be used in an industrial environment, as postulated in [75]. In this mixture, service-oriented principles should provide robust contract-based functionality specifications and straightforward interaction with higher layers of enterprise software. On the other hand,

the Publish/subscribe messaging mechanism should provide a modern and efficient way of dealing with the event-driven environments from the shop floor.

The SIRENA European project has pointed out the advantages of using SOA in industrial automation [76].

**Table 3 Effects on the use of service oriented architectures in automation [76]**

	Today	Near Future
<b>System</b>	Centralised, Large, Intelligent controllers, dumb devices	Decentralised intelligent & autonomous devices
<b>Communications</b>	Polling client-server point-to-point	Event-Driven, Publish-subscribe, peer-to-peer
<b>Setup</b>	Long and difficult, manual programming tedious debugging	No programming, plug and play, context-aware configuration

Robot manufacturers are also putting a great deal of effort into communications. An important standardization initiative is the XML-based Interface for Robots and Peripherals (XIRP)[77]. This effort is now entering its second generation, named XIRP+, which is closer to the service-oriented model.

The OPC *OLE for Process Control* foundation is moving their DCOM-based platform to the service-oriented model, with the OPC-UA (*Unified Architecture*) [78]. This architecture is built over WS\* standards and provides full interoperability with the OPC technology legacy. While this work is being written, there are neither many implementations, nor stable protocol specifications for OPC-UA.

Despite Microsoft's statement that *Microsoft Robotics Studio* (MSRS) also targets industrial automation, and the close relation existing with the industrial robot manufacturer *Kuka Roboter*, there are few visible industrial test-beds using this framework. Until now, the MSRS

platform's main use in industrial automation has been in education, using the physics engine associated with it to help students learn manipulator kinematics and dynamics.

## 2.8 Compiler technologies

The compilers' traditional use is to translate programming language into executable code, but their use is usually extended to analysers and transformation tools, among others. Throughout this work, several automatic generations of code will be used that require the use of compiler technologies. The general structure of a compiler is briefly introduced in the following paragraphs and the specific tools used in this work are detailed in section 7.3.1. These tools were used in two different software applications developed in this work.

In order to be scalable, modern compilers are organized in different phases that can be enclosed in two bigger groups: the **analysis**, which extracts structure and meaning from the original program, and the **synthesis**, which uses that structure and gives it a different form.

The **analysis phase** starts with the lexical analysis that divides the input stream into individual words called tokens, and is specified through regular expressions. In this stage, undesired things like white spaces are discarded, and the rest of the tokens catalogued. The syntax analysis parses the phrase structure of the program specified in the form of a context-free grammar. Abstract representations of the program are afterwards extracted, generally abstract syntax trees, and semantic actions performed over these representations to determine the phrase meaning, relating the use of variables with their definitions and performing type checking for expressions.

The **synthesis phase** starts with the intermediate code generation. The intermediate representation is a kind of abstract machine language without machine-specific dependencies and, furthermore, independent from the source language details. Intermediate languages uphold modularity with the separation of the compiler *front-end*, which includes the analysis phase and the intermediate code generation, from the compiler *back-end* which, starting from

the intermediate representation, optimizes the code for a specific platform and generates the machine code. A well-known example of an intermediate language is *Java ByteCode*.

## 2.9 Final note

From this brief overview, some final remarks should be made regarding assumptions and technologies used throughout the rest of the thesis. The use of **Ethernet technologies** (section 2.1.1) and **PC-Based architectures** (section 2.1.2) in industrial automation is now commonly accepted, and can successfully contribute to the reduction of costs associated with the use of mass market products. However, there are limits, especially concerning motion control or safety systems, and the coexistence of **two profiles of Ethernet** networking in the future is therefore foreseen: one relying on off-the-shelf Ethernet technologies originating in the mass market, and another with specialized hardware solutions. Similarly, it is predicted that PC-Based architectures will coexist with safe controllers depending on application needs.

The presence of Ethernet technologies increases the importance of the modern network integration technologies, like SOA. In this chapter, considerations on the different types of SOA, REST, WS\* Services and Device-Level SOA have been offered to guarantee some grounding knowledge for chapter 3. **Device-level SOA** is the obvious choice for use in the robotic work-cell because of the integration of event-based messaging mechanisms fundamental to industrial communications.

The concept of **reactive systems** has been introduced (section 2.5). The SME robotic work-cell, which is the main target of this work, is, in the device level, mainly a reactive system. The approach followed throughout the thesis takes this trait into account.

Compiler technologies have also been briefly introduced due to their importance in the development of the application presented in chapter 7.



# 3 SOA Middleware evaluation

---

In this chapter, a comprehensive evaluation of several service-oriented platforms is made regarding their use as a communication framework for manufacturing work-cells. Four different service-oriented platforms are described and two are selected to be tested against a real test-bed.

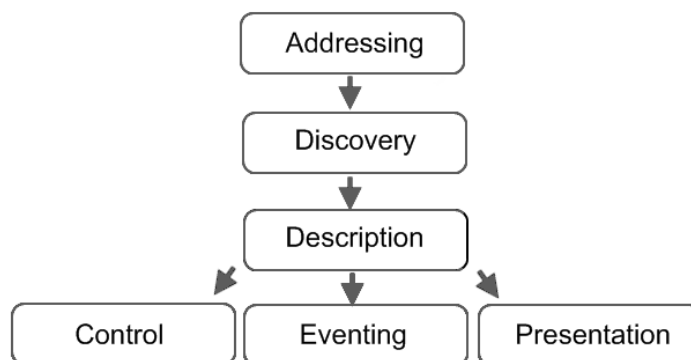
It is important to state that, because many of these technologies are hot-topics in research, both in the automation and in the internet/office world, there are several research groups trying to develop new middleware or improve the existing ones with the best features for given areas. It is not the case with this work, which will take the pragmatic approach of analysing existing solutions in order to retrieve important features for the problem at hand.

## 3.1 Universal Plug-n-Play

### 3.1.1 UPnP device architecture

The basic elements of an UPnP network are devices, services and control points. A device is a container of services and other devices. A service is a unit of functionality, that exposes actions and has a state defined by a group of state variables. A control point is a service requester. It can call for an action or subscribe an evented variable (variable with associated events). An

UPnP network's basic steps are shown in Figure 7 and briefly described in the following chapters.



**Figure 7. Different steps of an UPnP network**

Addressing occurs when a device or a control point obtains a valid IP address. The dynamic host configuration protocol (DHCP) is usually used; otherwise, the device or control point uses the auto-IP mechanism.

Discovery takes place once devices and control points are attached to the network and are properly addressed. The protocol underneath the discovery mechanism is the SSDP (*Simple Service Discovery Protocol*), that defines how devices should advertise their services and how control points search for devices in the network. An UPnP device can send two types of messages: **advertising multicast messages**, which are sent when the devices are added, renewing their advertisements or leaving the network; and **response unicast messages**, as an answer to control point requests. These messages (Listing 1), that use the NOTIFY verb described in the general event notification architecture (GENA) specification, are sent through a specific reserved port (1900) and include, among others, a pair of fields that together implement the lease concept: the unique identification number (USN) and the expiration time (max-age).

```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age = seconds until advertisement expires
LOCATION: URL for UPnP description for root device
NT: search target
NTS: ssdp:alive
USN: advertisement UUID
```

### Listing 1. Notification Message

Control Points send **broadcast searching messages** whenever they want to look for a device in the network.

The description step allows a control point to obtain the necessary information about a device. This is done through an HTTP getting request on the URL provided by the device in the discovery message. Inside an XML description message there are two distinct parts: the generic identification of device properties, usually called physical description, and the description of the services that are embedded inside the device, that is the functionality or logical description.

The first part (Listing 2) includes manufacturer and vendor information, like the model name, serial number, manufacturer URL, a unique device name (*UUID* namespace).

```
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
  <URLBase>base URL for all relative URLs</URLBase>
  <device>
    <friendlyName>short user-friendly title</friendlyName>
    <manufacturer>manufacturer name</manufacturer>
    <manufacturerURL>URL to manufacturer site</manufacturerURL>
    <modelDescription>long user-friendly title</modelDescription>
    <modelName>model name</modelName>
    <modelNumber>model number</modelNumber>
    <modelURL>URL to model site</modelURL>
    <serialNumber>manufacturer's serial number</serialNumber>
    <UDN>uuid:UUID</UDN>
    <UPC>Universal Product Code</UPC>
    <deviceType>urn:schemas-upnp-org:device:deviceType :v</deviceType>
```

### Listing 2. XML UPnP device description message – device properties

The second part (Listing 3) is a logical container of a list of services with all the information needed to interact with those services, namely: URLs for control and event subscription, the specification of the *serviceType* (fundamental information for the interaction with devices based on template descriptions, as seen on section 3.1.2) and the URL to get the SCPD (according to [79], this is an outdated acronym that used to stand for *Service Control Protocol Description*). An UPnP device can embed other devices, and their description is also embedded in this XML document right after the service list.

```
<serviceList>
  <service>
    <serviceType>urn:schemas-upnp-org:service:serviceType:v</serviceType>
    <serviceId>urn:upnp-org:serviceId:serviceID</serviceId>
    <SCPDURL>URL to service description</SCPDURL>
```

```

        <controlURL>URL for control</controlURL>
        <eventSubURL>URL for eventing</eventSubURL>
    </service>
</service>
...
</service>
</serviceList>
<deviceList>
...
</deviceList>

```

### Listing 3.XML UPnP device description message – service listing

The URL for presentation and some miscellaneous information regarding versioning and icon definition are also part of the device description.

At this stage, the control point has the necessary information about the actions and state variables provided by a device. The control step consists on action calling made by a device control point, and starts with the transmission of a SOAP message (Listing 4) from the control point to the device that includes the action name (in the format of an URN serviceType specification) and all the arguments to be used in this method calling.

```

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" s:
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:actionName xmlns:u="urn:schemas-upnp-org:service:serviceType:v">
      <argumentName>in arg value</argumentName>
    </u:actionName>
  </s:Body>
</s:Envelope>

```

### Listing 4. Message for UPnP method calling

The device answers with another SOAP message, which states the correctness of the response and the value of the return variables, either for the bidirectional or unidirectional arguments.

When the state of a service (modelled in terms of state variables) changes, the service publishes updates by sending messages over the network, following the format specified by the GENA. This mechanism is called *Eventing* and is materialized by 3 types of XML messages originating in the control point (*Subscribe*-Listing 5, *Unsubscribe*, *Renew subscription*), and 2 from the device (*Notify*, *Subscribe response*). The publisher keeps a table of subscribers and once a subscriber fails a renewal message, it is removed from it. A record from this table includes a unique subscription identifier, a URL for message delivery, and an *Event Key* that

counts the number of messages sent to that subscriber, providing in this way a checking mechanism for missing events on the subscriber's side.

```
SUBSCRIBE publisher path HTTP/1.1
HOST: publisher host:publisher port
CALLBACK: <delivery URL>
NT: upnp:event
TIMEOUT: Second-requested subscription duration
```

### Listing 5. Subscription message

The control step occurs whenever a control point wants to call an action in a device service. Invoking actions is a kind of remote procedure call, where the use of SOAP defines the use of XML and HTTP for remote procedure call. When the action is completed, the service returns results or errors. The service must respond within 30 seconds and the error should not be returned via an output argument.

Some devices may have presentation web pages. In this case, a control point can retrieve a page from the specified URL, load the page into the browser and, depending on the capabilities of the page, allow a user to control the device and/or view the device status.

## 3.1.2 UPnP Device Control Protocols (UPnP – DCP)

The UPnP device architecture defines a group of standard operations and a set of standard communication protocols that specify the message exchange patterns for operations like device discovery, description, and control. These operations are the building blocks on which actual UPnP devices are built, but they **do not define specific services and actions for devices to implement**. Even though a control point can use device discovery to determine exactly what services and actions a device supports, if there are no shared assumptions between services and actions on the same type of device, the control point can make no assumptions about the type of devices it controls. This means the control point must interact with devices of the same type as completely separate entities.

To address this issue, the UPnP framework proposes the *UPnP Device Control Protocols* (UPnP DCP), which allow a manufacturer to specify their service contracts according to a minimum set of requirements. For a networked CD player, p .e., the UPnP consortia has established a

template that defines six state variables as “required” (Table 4): *PlayMode*, *PlayProgram*, *DiscTOC*, *DiscNumberOfTracks*, *TrackNumber*, *TrackDuration*. However, making use of XML extensibility, there is space left for vendor specific variables.

**Table 4. State Variables for the UPnP PlayCD Template (adapted from [80])**

Variable Name	Req or Opt.	Data Type	Allowed Value	Default Value	Eng. Units
<b>PlayMode</b>	R	string	PLAY,PAUSE, STOP	STOP	n/a
<b>PlayProgram</b>	R	string	ONCE_IN_ORDER, REPEAT_IN_ORDER, ONCE_RANDOM, REPEAT_RANDOM	ONCE_IN_ORDER	n/a
<b>DiscTOC.</b>	R	string	(none)	(none)	n/a
<b>DiscNumberOfTracks</b>	R	ui1	>= 0, <= 255, += 1	(none)	n/a
<b>TrackNumber</b>	R	ui1	>= 0, <= 255, +=1	(none)	n/a
<b>TrackDuration</b>	R	ui1	(none)	(none)	ISO 8601
<b>Non-standard state variables implemented by an UPnP vendor go here.</b>	X	<i>TBD</i>	<i>TBD</i>	<i>TBD</i>	<i>TBD</i>

## 3.2 Device Profile for *WebServices* (DPWS)

DPWS is very similar to the *UPnP* framework. Both rely extensively on web technology and are close to the *WebServices* family. Although similar in many aspects, however, the UPnP and DPWS architectures use different languages for device description and different protocols for discovery and event notification (Table 5), which enables the DPWS initiative to provide a platform with protocols that are fully aligned with the *WebServices* technologies (both are developed inside W3C).

**Table 5. UPnP and DPWS protocol comparison**

	<b>UPnP</b>	<b>DPWS</b>
<b>Addressing</b>	<u>DHCP, AutoIP</u>	<u>DHCP, AutoIP</u>
<b>Discovery</b>	<u>SSDP</u>	<u>WS-Discovery</u>
<b>Description</b>	<u>UDA Schema</u>	<i>WSDL</i>
<b>Control</b>	SOAP 0.9, 1.1	SOAP 1.2
<b>Eventing</b>	<u>GENA</u>	WS-Eventing
<b>Presentation</b>	<u>HTTP, HTML</u>	<u>HTTP, HTML</u>

A proposal has been made to the UPnP foundation [81] for a convergence between the two approaches in the next major UPnP, but this version has not been delivered yet. According to [82], Microsoft's strategy for the future is to keep both technologies in parallel, *UPnP* focusing on home and entertainment devices due to their advanced stage of development in this market, and *DPWS* on office and company equipments, taking advantage of the vertical integration with the *WebServices* technology. During the launch of *Microsoft's* most recent operating system, *Microsoft Windows Vista*, both technologies were presented under the name *plug-and-play extensions for Windows*.

DPWS was designed taking into account lessons learned from the *UPnP* experience, namely in what concerns security and data contracts. Both these aspects had been neglected and postponed in the first *UPnP* specifications and that implied well-known problems. The appearance of the DPWS specification has generated great interest, especially given the success of the European project SIRENA [83], which pushed the development of several DPWS-related efforts with emphasis on several areas: industrial automation, telecommunications equipment, home automation and the automotive industry. From the rebound of this project, several others have appeared with narrower objectives: SOCRADES [72] focused on extending SIRENA with semantic knowledge, SODA [73] with providing tools for dissemination and engineering.

One of the major flaws of the DPWS in their current<sup>1</sup> standard is the absence of generic templates, similar to the *UPnP Device Control Protocol* presented in section 3.1.2. This problem has recently been addressed by Bobek in [84], and the resulting specification is expected to be integrated in the standardization/dissemination efforts being carried out by OASIS.

### 3.3 Decentralized Services Structure Protocol (DSSP)

The application model defined by DSSP results from the REST model, by exposing services through their state and a uniform set of operations over that state.

DSSP services are fine-grained entities that can be created, manipulated and destroyed repeatedly by DSSP operations, and their orchestration forms a DSSP application. A DSSP service consists of [74]: **identity** – global unique reference for the service; **behaviour** – definition of the service’s functionality; **service state** – current state of the service; **service context** – relationships the service has with other services.

An application is a composition of loosely-coupled services that through orchestration can be harnessed to achieve a desired task. DSSP achieves this by separating state from behaviour, and allowing services to expose their state and hide their behaviour. The behaviour of those services is described by contracts (that have a specific URI) that determine how a service can compose with other services. Such composition is called **partnering**.

To overcome some limitations of the traditional web-based architecture when applied to the device level, the HTTP/1.1 application model has been extended with structured data manipulation, event notification and partner management between services.

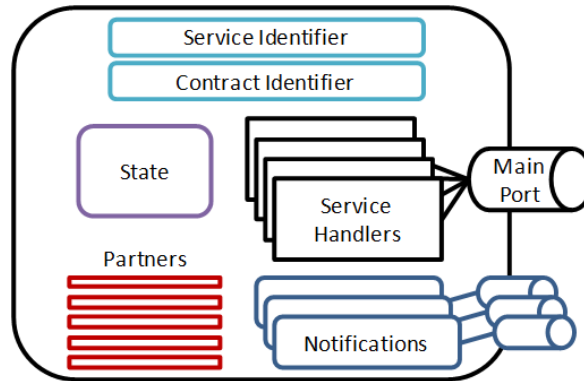
DSSP provides a uniform model for creating, deleting, manipulating, subscribing, and orchestrating services, independently of their semantics. DSSP defines a set of state-oriented message operations that provide support for structured data retrieval, manipulation, and event

---

<sup>1</sup> The evaluation of these platforms, especially the recent ones, is bound by the time this thesis is being written, July-December 2008.



notification. DSSP operations are an extension of the HTTP application model. The general structure of a DSSP service can be depicted in Figure 8 [85].



**Figure 8. DSSP service structure (Adapted from [85])**

### 3.3.1 DSSP service state

The state of a service is the representation of the service at any given point in time. Representing a motor may consist of rpms, temperature and fuel consumption. The messaging protocol underneath is SOAP based, and the state of a given service can be retrieved via the GET operation (Listing 6).

```
<MotorState xmlns:s="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:d="schemas.microsoft.com/xw/2004/10/dssp.html">
  <TemperatureC>23</TemperatureC>
  <RPM>1300</RPM>
  <FuelConsumption>4,5</FuelConsumption>
</MotorState>
```

**Listing 6. DSSP state**

The performance of the SOAP messaging is a recurrent issue in distributed platforms, due to the overhead of processing XML headers, which is very limiting especially considering systems with lots of sensor information, like those that *Microsoft* primarily targets with MSRS: mobile platforms. To address this issue, MSRS includes two optimizations for cases where distribution is not crucial [86]. In the first case, several services coexist in the same processing platform (either single-core or multi-core), which means that messages do not need to cross the network releasing the need for the SOAP header, even if the TCP encoding is still used. With this type of

internode communication, the MSRS platform reaches up to 5000 messages/s, against the 3000 msgs/s through the network [86]. If the communication occurs between services that are present in the same process and memory context, the TCP packaging is avoided, thus allowing rates of up to 100000 msg/s.

### 3.3.2 Ports, handlers and events

Every service has the *main port*, which serves as a gateway for all the messages that enter the service. This port is a typed port, which means that the messages accepted by the port are from well-defined types, including http 1.1 messages (GET POST), as well as DSSP-defined like *CREATE*, *DELETE*, *DROP* (see [74] for a complete list). Handlers are defined for each type of messages accepted by the main port. The management of timing and concurrency of handlers is the main use of the concurrency and management library shipped with MSRS (see section 3.3.4). DSSP services support the subscription of events by other services. Events are generated by the changes in the state of a service, and are posted in differentiated ports, i. e., each subscription received by a service will create a specific port.

### 3.3.3 Abstraction, contracts, partnership and manifests

The abstraction mechanism provided by the DSSP is based on **contracts**, which behave in a way similar to *UPnP* contracts, resembling the interface definition from object-oriented programming. The behaviour of a service (the contract) is the combination of the content model describing the state and the message exchanges that a service defines for communicating with other services. The behaviour of a service is identified by a globally unique URI known as the contract identifier. Abstraction and common functionality are achieved through the specification of generic contracts that describe basic functionality in the robotics field. Developers should build their services against these contracts in order to achieve reusability and provide a common functionality platform that sustains the development of more complex integrated systems.

**Partnership** of services is the mechanism used to safely specify service dependencies. A given service has a partner (service) whenever it depends on its functionality to work properly. Since

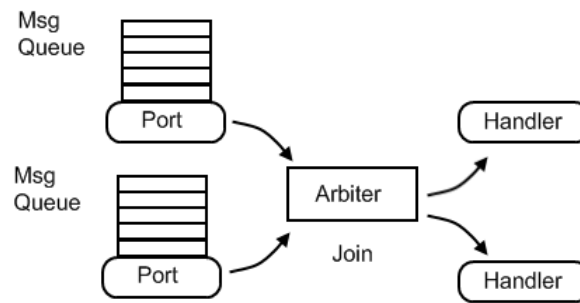
in loosely coupled architectures it is not possible to determine if a certain service is available, the partnership mechanism serves as a counter-contract mechanism, a guarantee that a service needs in order to provide functionality.

**Manifests** provide a way to create (not design) services declaratively. They consist of XML files that contain the information needed to start a service or a set of services, providing a simple way to define functionality across multiple services, i.e., they technically support service initialization and partnering. It should be noticed that complex compositions ought to avoid the use of manifests, and create the orchestrated services programmatically.

### 3.3.4 MSRS middleware

DSSP is tightly integrated in a middleware especially designed for robotics, here called the MSRS middleware. Although the current analysis focuses on the distributed platform, some important aspects of the middleware will now be analysed, since they are tightly coupled with the distributed platform and are very important for its success.

The primary objective of the MSRS is to provide a common framework for the development of robots, whose software usually contains many **concurrent operations**. The development of concurrent systems has always been very difficult, and the efficient (and safe) concurrent software development is nowadays still a very specialized activity. Coordination and Concurrency RunTime (CCR), a software library part of the MSRS platform [87], has been developed by addressing these problems, namely [88] coordinating between multiple operations, dealing with partial failure, and defining execution behaviour of asynchronous call-backs. To achieve this, CCR includes several modern abstraction mechanisms, which allow the user to program concurrently without the need to use semaphores or monitors. In a message-oriented environment (like MSRS), each service defines a queued port that is associated with a handler which processes that information. CCR provides the user with the power to define *arbiters* that can coordinate the behaviour of these systems in between the arrival of the messages and the handler calling. As an example, Figure 9 shows a join arbiter waiting for two messages to arrive from two ports, in any order.



**Figure 9. Join arbiter**

The definition of the arbiter also specifies which delegates (handlers) can safely run in parallel and which are mutually exclusive. The use of the CCR library already goes beyond the robotics field [89].

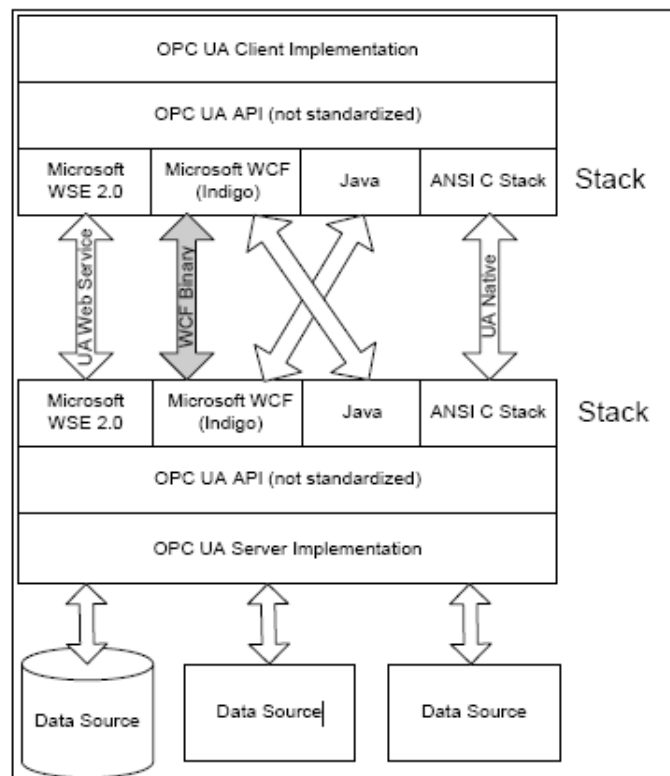
With relation to **simulation**, MSRS includes a powerful 3D simulator, which integrates seamlessly with the messaging environment, since the 3D entities integrate call-backs for dealing with events in the simulated world, like collisions, for instance. The 3D simulator not only includes the graphical engine but also a physics engine developed by Ageia [90], with the representation of forces, torques, friction and gravity updated at the frame rate.

MSRS middleware also includes a graphical programming language that, due to its importance to this work, will later be described in detail.

### 3.4 OLE for Process Control – Unified Architecture (OPC-UA)

Although the first announcements of the term OPC-UA date from 2005, at the time this thesis is being written, OPC-UA can be considered an upcoming technology. The basis for this statement is the slow and broadly discussed appearance of the standard that led to a specification in 2008 only and to the total inexistence of a programming stack for non-corporate members of the consortium. OPC-UA basically brings the powerful industry consortium OPC to the service-oriented wagon, with a specification that is fully compatible with the old DCOM-based OPC

standards [78]. The general architecture is presented in Figure 10 and shows that the OPC foundation proposes an additional layer (OPC UA API) that should be implemented over common platforms. The interoperability is therefore provided by the underlying platforms, like *Java* and *WCF WebServices*, but compromised if the programmer chooses to use the ANSI C Stack.



**Figure 10. OPC UA Architecture- Adapted from [78]**

To address the performance problems related with the use of XML-based messaging, OPC-UA defines a binary encoding called UA – Binary. This encoding type can be used over HTTP in order to provide firewall crossing, or directly on the wire, thus providing very fast network access times.

In terms of functionality modelling and leveraging, OPC-UA defines a very strict step of standardized meta-model, providing generic UA clients with good access to data. OPC UA

Information Model Standards define the extension mechanism that permits vendors to expose extra functionalities.

### 3.5 Platform choice

The choice of the platforms to be evaluated against a real test-bed was conditioned by several reasons, not only due to technical/architectural features of the middleware but also to practical implementation reasons. As stated before, it was not the intent of this work to design new middlewares or to explore new architectural styles. This derives from the belief that available and extensively discussed networking architectures would be a good platform for research.

The choice of the platforms to be tested against the real test-bed was ruled by **two** criteria: architectural style and extensibility mechanism.

The **architectural styles** can be divided into two large groups. On one side we can find the platforms closely related with the WS\* technologies, DPWS, UPnP, OPC-UA, and on the other a DSSP with its *restful* approach.

Interesting comparisons can be made between the **extensibility mechanism** for the *UPnP*, *OPC-UA* and the *DSSP* platforms. All these platforms provide infrastructures for the device definition, but the standardization process is taken on in different ways. The UPnP mechanism is freely available to the developer for private use, but the standardization stays with a forum, that evaluates current implementations and decides which common functionality represents a standard. The DSSP mechanism enhances the detail of each specification, leaving the definition to the platform promoter. Nevertheless, it seems that Microsoft will promote the definition of common functionality from the community (non-formally defined). For the OPC foundation, a strict top down mechanism is used to define what is or is not basic functionality, which nowadays is closely connected with former OPC-DA (Data Access) models. The definition of extra functionality is decided by the vendors, but since OPC will keep backwards compatibility, it is rather unlikely that any of these specifications ever reach the standard. The DPWS

middleware does not define any standardized mechanism for template extensibility, which configures a known issue that Bobek has very recently addressed [84]. This proposal is nevertheless not yet integrated in available DPWS stacks and, furthermore, at the time the choice was made there was not even a proposal. In terms of choice, high importance was given to the mechanism of extensibility, due to the fact that it is a cornerstone in the development of automatic reasoning over service-oriented platforms (see section 4)

It should be noticed that the choice of the platform was made at the beginning of this work. In this timeframe, the OPC-UA specification was a late arrival (available programming stacks are not yet available) and there weren't any programming stacks available for DPWS until 2007. Furthermore many interesting features, like the extensibility mechanism, are still missing today.

The evaluation of these criteria has resulted in the choice of UPnP and DSSP for the middleware comparison.

UPnP represents the traditional group of service-oriented platforms, and provides several guarantees in practical issues, like the availability of SDKs and the completeness of the platform. Furthermore, due to the similarities between *UPnP* and *DPWS*, most conclusions can be shared by both platforms.

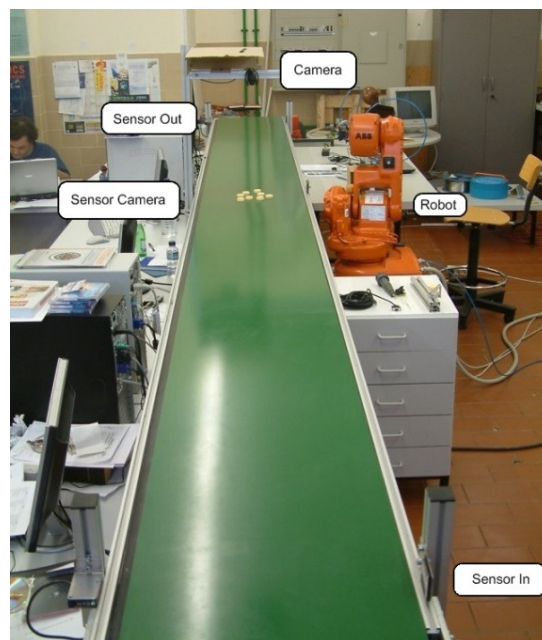
DSSP represents a different paradigm in terms of device level service-oriented platforms with the introduction of restful services. Besides, DSSP is integrated in a complete framework with very interesting features, namely the support for visual programming and the extensive concurrency support.

### 3.6 Case study- test-bed description

Robotic work-cells for SMEs have special requirements, and an experimental setup had to be mounted in order to evaluate them. The basic requirements were the presence of modern sensing systems, like p.e. a vision system, extensive use of human-machine interfaces and the use of available technologies (out of the box PLCs, robot controllers).

The goal was to evaluate the use of SOA as middleware for the industrial robotic cell and to compare it with traditional approaches to the same problem. Previous work made over the same laboratory setup [91] constituted a good comparison basis. In [91] Ethernet network to manage communications between devices was used, defining an ad-hoc messaging scheme over TCP/IP sockets. This solution presents many problems that device-level SOA should address, namely the messaging scheme being proprietary, the absence of discoverability mechanisms and the solution being tied to specific IP/addresses and ports.

The robotic cell used in this demonstration is composed of an ABB IRB 140 robot, equipped with the latest IRC5 controller, a conveyor controlled by a Programmable Logic Controller (PLC) (Siemens S7-200) and a vision system. For the control of the cell, a PDA (Personal Digital Assistant) and a voice recognition system are available. In general terms, the conveyor transports sample pieces over the machine's vision system (Figure 11), which calculates the number of pieces and their corresponding position. An operator makes a visual inspection of the available pieces and chooses some of them to be taken out by the robot. The instruction from the operator should be made by voice commands or via the PDA interface.

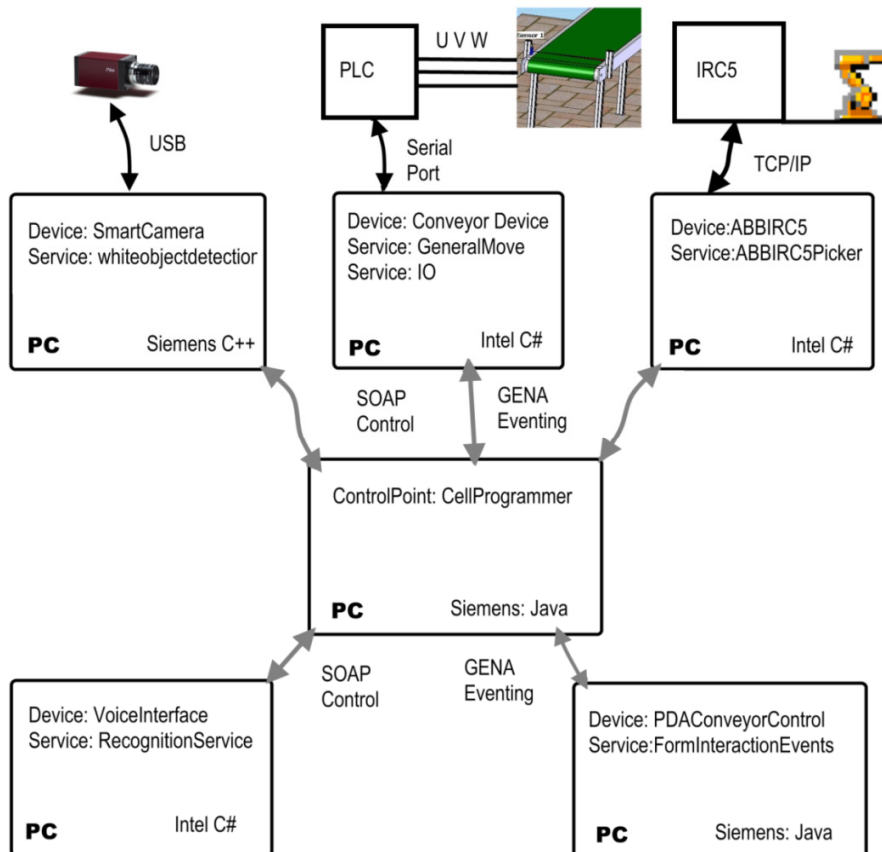


**Figure 11 Experimental setup at the laboratory.**



### 3.7 UPnP setup

The general communications architecture based on the UPnP technologies is presented in Figure 12.

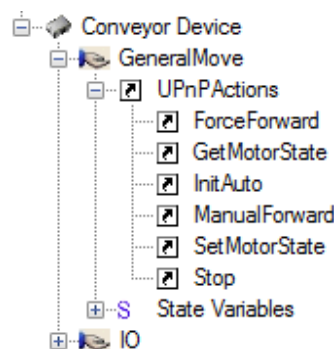


**Figure 12. General UPnP architecture**

This architecture is composed of a network of UPnP devices which contain several services that can be orchestrated via a control point. Since the PLC and the industrial robot have limited support for the development of software for the embedded processor, the need arose to develop a layer of several PC-based applications to distribute their services over the network. Some devices are counterparts of the hardware components, like in the PLC or the robot controller, but others are units of functionality without any hardware counterpart, like the voice or PDA interface. The separation of the cell orchestration and the interface (voice or

graphical) is very uncommon and represents the application of service-oriented patterns, where services are related to units of functionality that are not necessarily related to hardware features. To demonstrate this functionality approach, two different services have been developed for the conveyor, one more device-related and the other more process-related.

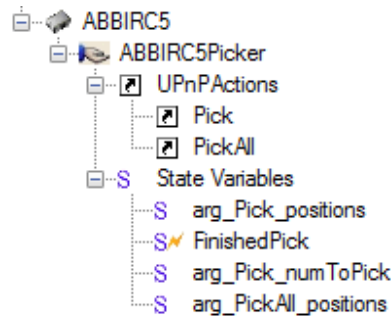
The Conveyor UPnP device was implemented as a software application that communicates with the conveyor, commanding the PLC through a serial port. As mentioned above, two services were developed (Fig. 4). The *HighLevel Prog* service provides actions and variables with process-related meanings, whereas the *Maintenance\_Setup* service is more technology-related, and is intended to be used during the development or maintenance stages. This strategy of dividing process-related and technology-related services enhance the advantages of the SOA in the HighLevelProg service, without compromising the technological know-how needed for finding IO problems in the installation stage, for example. This software application integrates the Intel C# UPnP stack [92] as communications infrastructure.



**Figure 13. UPnP device for the conveyor**

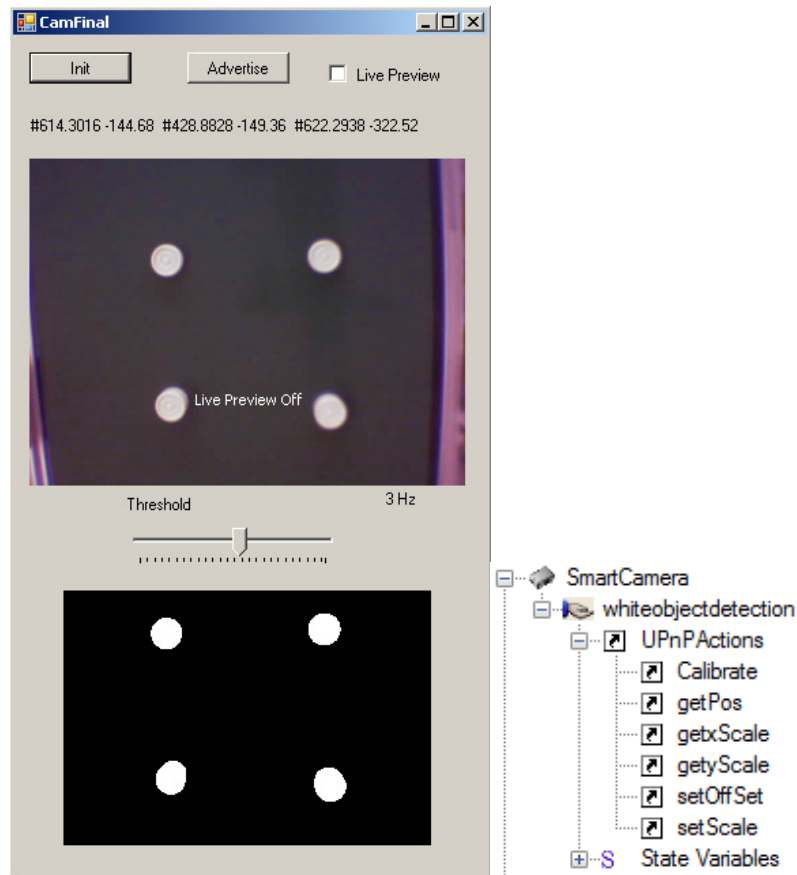
The RobotIRC5 UPnP device was implemented in a software application that communicates with the robot controller via a TCP/IP network. The robot controller runs a server application developed in RAPID [93] as an independent task, similar to the one presented in [91]. This UPnP device provides one service: *PickAndPlaceService* (Figure 14). This service has two different actions: one that allows picking all identified pieces, and another that picks a properly parameterized single piece. It also includes an evented state variable (*FinishedPick*) that

indicates the state of the robot, and publishes an event each time the robot finishes picking. The software application that implements this device also uses the Intel C# UPnP stack.



**Figure 14. UPnP Device developed for the industrial robot**

The detection of the position of pieces over the conveyor is provided by a PC with a commercial USB web camera and a customised software application developed using C++ together with the *Microsoft's UPnP* stack (C++/COM) [94] and the Intel OpenCV vision libraries [95]. This software consists of a simple vision appliance that extracts blobs from a frame and calculates their positions in the robot's coordinate system. This application includes a graphical interface that allows the user to adjust the threshold value and verify some data from the system, like the currently detected positions.



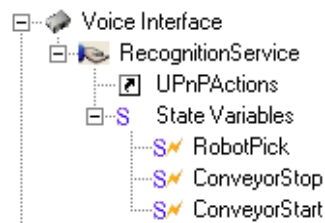
**Figure 15. UPNP device developed for the Smart Camera and graphical interface**

The SmartCamera UPNP device contains services that expose the functionality of the vision system (Figure 15). The `getPos` function allows an UPNP control point to retrieve the current position of pieces.

In this device, there is no separation between the setup functionality and the operation functionality, like the one verified in the *Conveyor* device. Nevertheless, there are some functions thought to be used during setup stages, like *Calibrate*, that provides the user with the ability to parameterize the transformation between the robot coordinate system and the camera coordinate system by using 4 pieces in front of the camera and introducing the robot position for the same pieces.

For the voice interface, a service was developed with UPNP state variables associated with recognition sentences that match operation commands (Figure 16):

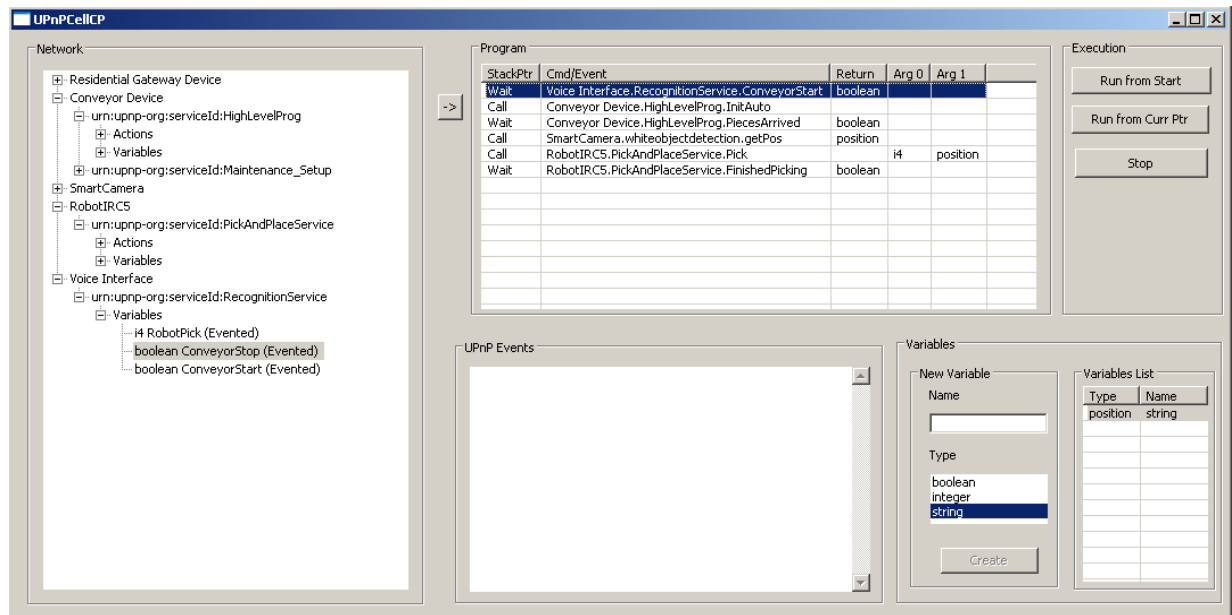
- Conveyor Start
- Conveyor Stop
- Robot Pick <number>



**Figure 16. UPNP device developed for the voice interface.**

For the first two sentences, a *boolean* variable was associated with each command, and each time the sentence is recognized, the system fires an UPNP event. The third command includes some extra information (the number of the piece to pick), which implied the use of a double variable associated with the recognition event. The use of voice recognition systems in industrial environment and their integration in Service Oriented architectures is later discussed in section 5.3.

The **Cell Programmer Interface** (Figure 17) is a software application created to control the flow of high-level tasks in a manufacturing cell. It was developed using the *Siemens Java UPNP* programming stack [96] and is basically an UPNP control point, with some tools suitable for the building of a generic stack. This stack represents the control flow of process-related tasks. On the left side of this interface, a tree shows all UPNP devices present in the network (notice the presence of a “stranger” gateway device). By clicking over them, it is possible to get additional information (access the presentation page, for example). Using the “arrow” button, actions or events are added to the stack. Furthermore, when running the resulting program and the program counter is pointing to an event, it means that the program is “waiting” for that event to occur. Inversely, if the program counter is pointing to an action, it means that it is calling that action and waiting for the return. There is also the possibility of defining auxiliary variables to store values that can be used as arguments in later stack steps.



**Figure 17. UPnP control point: Cell Programmer Interface**

The simple case depicted in Figure 17 (Program Group Form) represents the proposed pick-n-place operation used to evaluate the SOA platform. In this program, the setup should wait for a speech recognition event that commands the conveyor to start. When the event occurs, an action is called commanding the conveyor to enter automatic mode. The next UPnPAction is called to obtain information about the number of pieces and their respective positions from the camera server. After this, the system waits for the voice command of the operator regarding the number of pieces to pick, and with this information, combined with the one that the camera service provides, it calls the robot action *Pick* for the robot to take the pieces off the conveyor. The end of this operation triggers the event *FinishedPicking*, the system waits again for the initialization command and everything restarts. A video of this application can be seen in Video 1 (Appendix B).

### 3.8 DSSP setup

The implementation of DSSP was made with the objective of allowing a precise comparison with UPnP's counterpart: the MSRS package. Consequently, 3 services were developed which

resemble the UPnP services: *ConveyorMRSR*, *Abbirc5*, *SmartCam*. All these services were developed using the C# programming language from scratch, rejecting any of the MSRS supplied services. This enables a more precise comparison, since it is done using similar services (built in the same way). Nevertheless, one exception has been allowed for the *VoiceUPnP* service. The speech recognition was developed using the MSRS-provided speech recognition service, and the recognition logic programmed using the *Microsoft Visual Language Programming*-MVLP. This also allows the evaluation of MVPL, and highlights the benefits of the reuse of services when they are integrated in a common middleware.

MSRS services are specified by contracts. These contracts specify which message ports and which type of messages are available. Like any RESTfull approach, the DSSP operations involve exchanging messages of specific types, which in this case are: CREATE, DELETE, DROP, GET, INSERT, LOOKUP, QUERY, REPLACE, SUBSCRIBE, SUBMIT, UPDATE and UPSERT. For the robot, the contract is shown in Figure 18.

When comparing this service with the one presented in the UPnP implementation, the obvious conclusion is that the *UpdatePick* operation is not a real Update operation but a way of managing to emulate the *Pick* method. On the other hand, the *UpdateMotorState* is a nice substitute for the UPnP *MotorOn()* and *MotorOff()* methods.

Abbirc5

The Abbirc5 Service

Name	Abbirc5Service
Display Name	Abbirc5
Description	The Abbirc5 Service
Contract	http://schemas.tempuri.org/2007/09/abbirc5.html
CLR Namespace	
Assembly	ABBIRC5.Y2007.M09.dll
Service Prefix	/abbirc5
Initial State	

Partners

http://schemas.microsoft.com/xw/2005/01/subscriptionmanager.html

Main Port

Name	Robotics.Abbirc5.Proxy.Abbirc5Operations
------	--

Operations

Lookup <sub>DsspDefaultLookup</sub>	<b>Type:</b> Microsoft.Dss.ServiceModel.Dssp.DsspDefaultLookup <b>Request:</b> Microsoft.Dss.ServiceModel.Dssp.LookupRequestType <b>Responses:</b> Microsoft.Dss.ServiceModel.Dssp.LookupResponseType W3C.Soap.Fault	Get <sub>HttpGet</sub>	<b>Type:</b> Microsoft.Dss.Core.DsspHttp.HttpGet <b>Request:</b> Microsoft.Dss.Core.DsspHttp.HttpGetRequestType <b>Responses:</b> Microsoft.Dss.Core.DsspHttp.HttpResponseType W3C.Soap.Fault
Drop <sub>DsspDefaultDrop</sub>	<b>Type:</b> Microsoft.Dss.ServiceModel.Dssp.DsspDefaultDrop <b>Request:</b> Microsoft.Dss.ServiceModel.Dssp.DropRequestType <b>Responses:</b> Microsoft.Dss.ServiceModel.Dssp.DefaultDropResponseType W3C.Soap.Fault	Submit <sub>HttpPost</sub>	<b>Type:</b> Microsoft.Dss.Core.DsspHttp.HttpPost <b>Request:</b> Microsoft.Dss.Core.DsspHttp.HttpPostRequestType <b>Responses:</b> Microsoft.Dss.Core.DsspHttp.HttpResponseType W3C.Soap.Fault
Get <sub>Get</sub>	<b>Type:</b> Robotics.Abbirc5.Proxy.Get <b>Request:</b> Microsoft.Dss.ServiceModel.Dssp.GetRequestType <b>Responses:</b> Robotics.Abbirc5.Proxy.Abbirc5State W3C.Soap.Fault	Update <sub>SetMotorState</sub>	<b>Type:</b> Robotics.Abbirc5.Proxy.SetMotorState <b>Request:</b> Robotics.Abbirc5.Proxy.SetMotorStateRequest <b>Responses:</b> Microsoft.Dss.ServiceModel.Dssp.DefaultUpdateResponseType W3C.Soap.Fault
Update <sub>StopProgram</sub>	<b>Type:</b> Robotics.Abbirc5.Proxy.StopProgram <b>Request:</b> Robotics.Abbirc5.Proxy.StopProgramRequest <b>Responses:</b> Microsoft.Dss.ServiceModel.Dssp.DefaultUpdateResponseType W3C.Soap.Fault	Update <sub>RunProgram</sub>	<b>Type:</b> Robotics.Abbirc5.Proxy.RunProgram <b>Request:</b> Robotics.Abbirc5.Proxy.RunProgramRequest <b>Responses:</b> Microsoft.Dss.ServiceModel.Dssp.DefaultUpdateResponseType W3C.Soap.Fault
Update <sub>Pick</sub>	<b>Type:</b> Robotics.Abbirc5.Proxy.Pick <b>Request:</b> Robotics.Abbirc5.Proxy.PickRequest <b>Responses:</b> Microsoft.Dss.ServiceModel.Dssp.DefaultUpdateResponseType W3C.Soap.Fault		

Figure 18. Abbirc5 Contract

The contract information is available via HTTP since the service is running, and the state can be retrieved via the HTTP get operation (Listing 7).

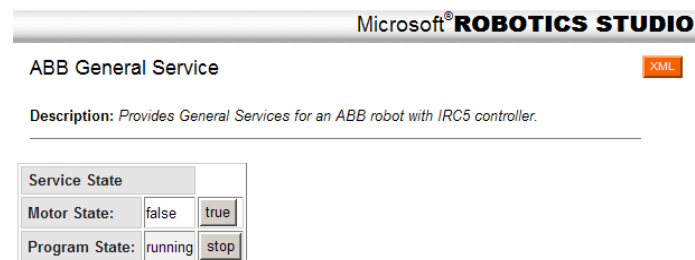
```
<Abbirc5State xmlns:s="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:d="schemas.micorsoft.com/xw/2004/10/dssp.html">
  <ProgramState>running</ProgramState>
  <MotorState>>false</MotorState>
</Abbirc5State>
```

Listing 7. Abbirc5 state

The interaction with the services in setup/manual mode is an important feature for industrial systems. To improve interaction with the services, an eXtensible Stylesheet Language Transformations (XSLT) transformation has been developed. This transformation gives the html look (Figure 19) to the XML state and provides a user interface to update the service state in a similar way as the generic control point in the UPnP setup. It is important to notice that it is not



possible to develop a generic interface for manual interaction with the services MSRS. This is due to the dependence between the interaction and the message types. On the other hand UPnP relies on common standard data types and specifies the type of interaction available (Action Call or Event subscription) supporting the use of generic control points for manual interaction with their services.



**Figure 19. Abbirc5 XSLT/HTML interface**

The orchestration in the DSSP has been implemented using the MVPL (Figure 20). In this orchestration, it is possible to see the semantic retrieving from the data raised by the speech recognition service of the MSRS platform. This diagram renders the dataflow nature of the MVPL, with events raised from state changes, like the recognition of a voice command, flowing through services to produce the desired output. The diagrams are clear to interpret, although the reasonable amount of C# code that was needed inside some services to get the expected behaviour out of each port should be noticed. It should also be noted that this programming takes advantage of the dependence of the underlying platform, *.Net Framework*, allowing the user to use casts to shortcut conversions (see the conversion of the voice command), and furthermore the use of complex data types (see *modeConveyor*).

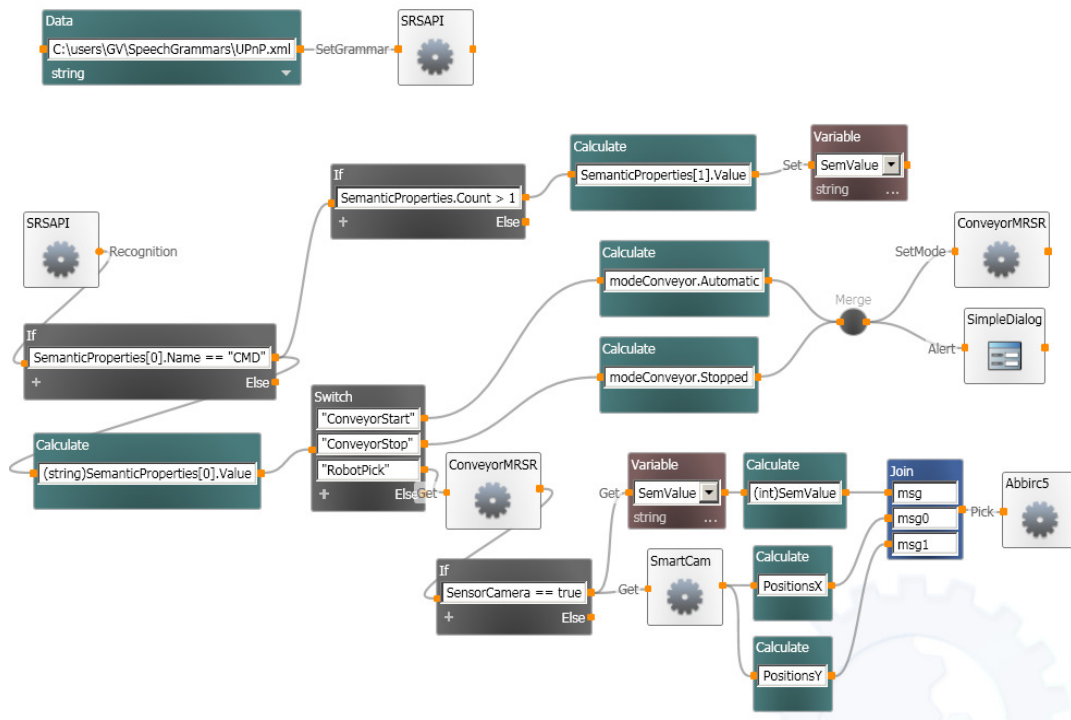


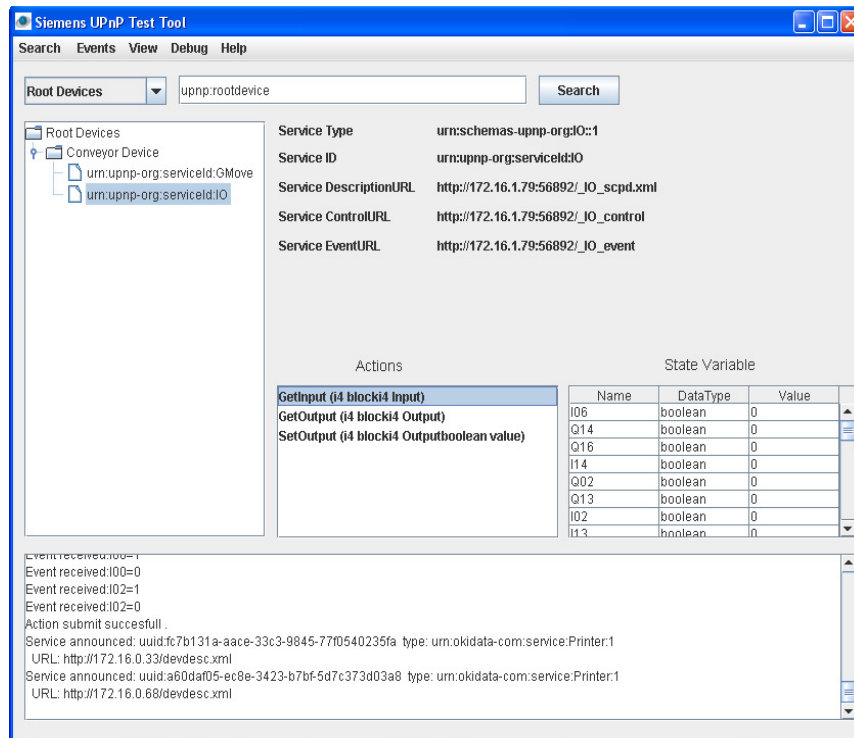
Figure 20. Orchestration using Microsoft Robotics Studio

### 3.9 UPnP/DSSP comparison and conclusions

Experiments using UPnP and DSSP with the industrial test-bed have provided valuable information that can be used to select the SOA best suited to become a platform for industrial robotic cell programming. The evaluation has been essentially qualitative and based on feedback provided by engineers and robot programmers from systems integrators, as well as graduate students with or without previous experience with orchestration techniques. The proposed comparison considers two main topics: the **architectural style** is discussed in section 3.9.1 and targets the comparison between the REST and the WS\* style of web services; the evaluation of the actual **middleware platform**, in section 3.9.2, lists specific platform features with interest for industrial robotic cell environments.

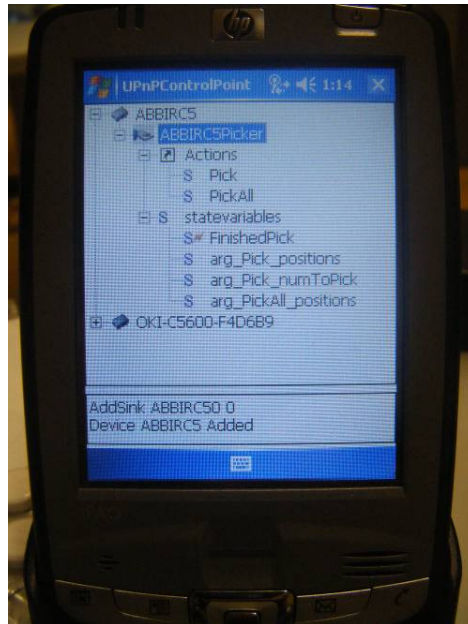
### 3.9.1 Architectural style comparison

The architecture styles differ radically in DSSP and UPnP. This is particularly visible in MSRS since there are no RPC-like methods in Microsoft's DSSP. In this architecture, everything consists of messages, and everything is driven by (or driving) state changes. This is an immediate consequence of the RESTful flavour of the architecture. It is obvious that every RPC call can always be replaced by a specific message and most of the times by a CRUD message (Create, Retrieve, Update and Delete), without any need for creating a specific one. The question is whether this model is the best way to express what the programmer has in mind, and whether the actual procedural programming model of most industrial robots will cope well with the REST style. Consider for example the action "Pick" from the *Abblrc5* from the UPnP setup. This method represents the operation of picking pieces and has as arguments the position of the pieces and the number of pieces to pick. This operation does not fit in any of the DSSP message operations and the update-message used seems like an unnecessary workaround. Recalling the discussion on message patterns in section 2.5, the RPC concept has evolved and is no longer related to the blocking of a remote call. *UPnP* is a good example of this, since its procedural calls can be either synchronous or asynchronous. Nowadays, the RPC is a programming abstraction more related with the way messages are specified. Consider for instance the interaction needed with available services and functionality during the development stage: Figure 21 shows the interface of a generic *UPnP* control point that can be compared with the state of the DSSP service shown in Figure 18. To obtain a friendly interface (Figure 19) for the DSSP service, an XSLT transformation layer needed to be developed, which must be done for each specific case. Due to the simplistic nature of the specification of services via methods (or operations), the development of generic clients that can act as maintenance or setup tools is very simple. In fact, all major UPnP provide a generic control that can perform this role, like the *Siemens* shown in Figure 21.



**Figure 21. Generic UPnP control point**

The enhanced usability provided by generic control points led to the development of a generic control point for a portable platform, a PDA (*Personal Digital Assistant*) (Figure 22). This application was developed for Windows Mobile 6 [97] using the .Net Compact Framework [98] platform. Due to the inexistence of UPnP programming stacks for this platform, the development of this software included the development of a stack derived from the one from Intel [92]. The PDA generic control point revealed very useful during the daily use of the workcell.



**Figure 22. PDA Generic Control Point**

### 3.9.2 Platform comparison

The following discussion is based on the experience gained from implementing both architectures with the presented test-bed. The idea here is to point out the main differences and highlight the interesting features of each technology with special attention to **messaging patterns, language/platform independence, concurrent programming, service discovery, security, visual programming features** and **integration with current technologies**.

Important care should be taken when comparing these platforms in terms of **messaging patterns**. Although UPnP and DPWS are related to the Web Services world, the debate over messaging, known as REST versus SOAP, is not valid when analysing device-level SOA, because RPC is not the sole interaction mechanism. UPnP and recently DPWS (with the introduction of *WS-Eventing*) provide publisher/subscriber mechanisms that are message-based interaction paradigms.

In terms of **language/platform independence**, UPnP takes the lead. Built over common and public standards, there are toolkits available for every major platform (Windows, Linux, MAC) and language (C#, Java, C++, Rubi). On the other hand, Microsoft Robotics Studio relies

exclusively on the .Net platform, and is inherently limited not only to Microsoft languages but also to .Net data types, which can represent a barrier to wider adoption. Moreover, many of the important features of the platform, like the discovery of services or the management of distributed messaging are closed inside *dlls*, which limits its scalability.

**Concurrent programming** is very important in industrial automation. Even when using coarse-grained services, as stated before, the availability of powerful tools (library, SDK...) to help the deployment of concurrent programs is very important. Although many work-cell orchestrations can be described via a simple stack structure like the one presented in UPnP experiment, more complex examples will ask for better concurrency support. In this aspect, the UPnP platform is limited and asks for complementary concurrent features (semaphores p.e.) from an existing library (*Lund Java-based Real-Time Library* would be a suitable example [99]). Together, DSSP and CCR constitute the core of the MSRS runtime and were designed with concurrency problems in mind. This means that DSSP have extensive support for concurrency, which is extended to the graphical programming level.

The UPnP service **discovery** is processed on a peer-to-peer basis. Every control point has the ability to discover devices. The basic unit of discovery is the message, which makes the system less coupled. In Microsoft Runtime, services can discover each other through a simple discovery service that acts as a rendezvous point between services running on a runtime and between runtimes. This can become a problem, since a failure in the discovery service may stop the discovery mechanism. On the other hand, the discovery mechanism of the MSRS middleware can find services that are not running. The partnership mechanism between services relies on this feature to wake a service when it is needed.

Although many industrial networks are divided from the outside office network, it is always relevant to have **security mechanisms** in the SOA platform. UPnP does define a security mechanism [100], but its specification arrived late, which has led to many proprietary security protocols, and to the absence of security in many programming stacks. The DSSP runtime has a set of infrastructure services to deal with security issues. This solution is better than the one presented in UPnP because once you have the runtime you have standard security.

**Visual programming** was one of the key elements in the framework evaluation. The idea of transparent networking through service-oriented middleware and a high-level programming language for the service orchestration convinced all the system users. In this particular aspect, the *Microsoft Visual Programming* environment revealed itself to be very friendly and appealing. On the other hand, the programming principle behind the visual programming language, the data flow programming, revealed itself less appealing to the user than the stack of events and calls shown in the UPnP test case.

One of the things that an SOA should be is a suitable platform for the development of **HLP features**. One of the keys for HLP is environment sensibility, which is easily reached with a “hot plug-and-play” discovery, as we found in UPnP. In this scenario, UPnP-discovery (peer-to-peer) takes advantage. On the other hand, the mechanism that allows the composition of services provided by DSSP (called partnering) seems a very powerful way of defining dependencies between services. Considering the High-Level-Programming perspective, these dependencies are requisites for a service to run.

Microsoft Robotics Studio intends to be an end-to-end solution for robotics and there is an enormous dynamics in developing new services with very interesting features. Services like speech or hand gesture recognition are shipped with the main installation of the runtime. These tools enormously facilitate the deployment of industrial orchestration applications based on building blocks, and many of them are useful to industrial cell programming. On the other hand, UPnP does not have similar tools available.

### **Integration with current technologies**

An important issue to consider regarding the industrial automation integration is the specification of services. Most industrial robots still define their tasks by means of a procedural language (object-oriented in some cases). These languages seem very adequate for the specification of services to the network, using an approach similar to the one used with Web Services (SOAP/WSDL) and languages like Microsoft Visual C# or Sun’s Java.

### 3.9.3 Conclusions

The main objective of this chapter has been to review some of the recently proposed SOA technologies, and to confront the most promising approaches with experimental setups reflecting real applications, i.e., using the selected SOA architectures to control a real manufacturing cell and evaluating the results. Focusing on industrial automation and specifically on industrial robotics cell programming, SOA can enable automation engineers to focus on their expertise (machine vision, force control, etc.), allowing them to keep their favourite platform/language, to rely on standard technologies, and to reduce their attention to the interconnection tricky tasks.

For several months, this system shared the same test-bed with other distribution technologies and, in comparison, the ubiquitous presence of services combined with easy access to **generic clients**, like the ones present for the UPnP setup, proved to be a very valuable aid in terms of **maintenance and system setup**.

Programming industrial robotic cells using SOA for a framework, and friendly high-level **graphical interfaces** for specification of the system logic proved to be less time consuming than traditional object-oriented techniques applied against a similar setup [91]. Furthermore, the opinion of users of the **two-layer structure** (SOA and orchestration) was very positive, which bound the rest of the research, as described in the following chapters.

Both technologies tested were not specifically designed for usage with cell integration. UPnP is more suitable for home automation (especially the media-rendering profiles), and DSSP from Microsoft Robotics Studio seems more suitable for fine-grained services (typically found inside a mobile robot: for example, controlling a motor with events from sensors), which means that it is not easy to reuse services that were not developed with the robotic work-cell scenario in mind. This enhances the **problems related with the integration of SOA technologies with the current robotic automation**, especially in SME's, whose reality cannot be compared with the IT (Information Technologies) world, where a significant numbers of programmers is available.



# 4 Proposed solution

---

This chapter introduces the solution proposed in this work for the issue of programming an industrial robotic cell applied to SMEs. The proposed solution is based on a wide number of knowledge inputs from the analytical evaluation of SME needs (chapter 1), on technical knowledge related to the evaluation of service-oriented platforms (chapter 3), and also on the author and his laboratory's previous experience with industrial robotic cells in SMEs ([45][46][47][48]), as well as on the feedback obtained from joint cooperation inside the European project SMERobot™.

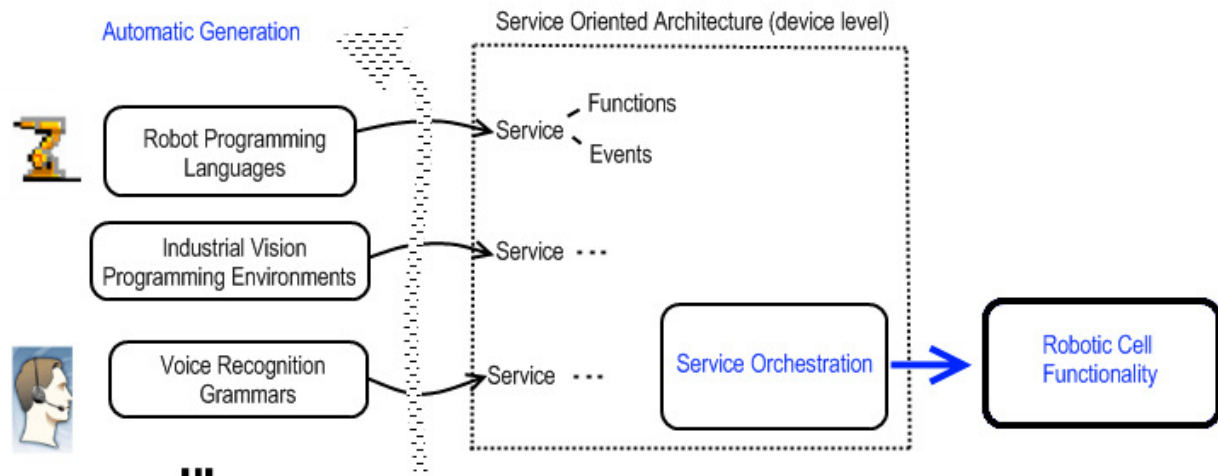
The evaluation of service-oriented middleware platforms presented in chapter 3 has shown that their use can be very valuable to the acceptance of robots in SMEs. However, some critical issues were pointed out regarding the adoption of these network platforms, namely, who will **program those services**, and how can they be **orchestrated (integrated)** without programming in Java or C++.

Another important aspect is the **simplicity** of the integration process. In this work, the target of the use of SOA is industrial robotics. Considering the past history of industrial automation programming languages, one can say that simplicity is very important, and special attention should be paid to the person who is going to ultimately use and maintain the system. PLC programming is a good example of how important the match between the language and the language user is. Ladder diagrams were introduced in the 70's as a PLC programming language and its logic structure was inherited from relay circuits, in order to obtain acceptance from the engineers at that time. Its success throughout the years proves that simple dedicated

programming languages can be very adequate for industrial systems. Furthermore, the ladder language has survived the coexistence with several languages [101] that compose the IEC61131 standard [102], which purposed to address the limitations of the ladder language.

The integration of service-oriented architectures in SME robotic work-cells can be framed in different approaches. One approach is the integration within knowledge-based configurable working as middleware platform for plug-n-produce. The combination of semantics with service-oriented architectures is a promising research topic that has pushed forward several research efforts [34][103][104] [105], including the EU project SOCRADES [72]. This is a long-term perspective that will explore the benefits provided by *device* and *service* templates present in most modern SOA middleware platforms. These profiles were described in section 3.1.2 and allow suppliers to agree on common functionality, which enables a consumer to design applications targeting generic services, regardless of their detailed implementations. In these scenarios, a robot user could p. e. plug a 2D vision system, a robot and a drilling machine into the work-cell network switch and, since all these devices comply with generic templates, a software application should be able to reason about the functionality provided by the complete set. As a result of this reasoning process, this application would propose the functionality of drilling holes in previously cross-marked places to the user, without any programming or networking knowledge. Although the results from these works are promising, the perspectives for usable platforms are only **long-term**. This reality is further enhanced by the fact that this approach is very dependent on standardization efforts. The most significant standardization effort that can lead to work valuable in the robotics field is being developed by a European consortium of robot manufacturers with the XIRP [77]. It is also important to note that, regardless of the evolution of the reasoning systems, these are expected to be very close to application types, like for example welding or gluing, thus exploring the synergies present in the actual industry panorama. In the short term, this fact limits the flexibility of the solution and goes against one of the premises discussed in the introduction to this work (section 2.2.1), related to the use of commercial off-the-shelf human/machine interfaces that are usually designed for the office or home environment.

A different **short-term** approach presented in this work relies on the flexibility of automatic generation techniques, like modern compiler technologies and programming environments, to integrate devices using current state of the art technologies. The general architecture of this solution is presented in Figure 23.



**Figure 23. General Architecture**

In this approach, the simplicity of the system is a major goal.

In terms of **service programming**, this alternative purposes the use of current technologies, either in terms of programming languages or programming environments, in an attempt to bridge the gap between the programmer and the technology. In chapter 5, two examples are presented that demonstrate the automatic generation of services from voice recognition and robot programming grammars.

The success of **domain-specific languages** (DSLs) in industrial automation, together with the conclusions retrieved from the SOA evaluation carried out in chapter3, point to a domain-specific graphical programming language as a powerful way of **orchestration**. Therefore, this solution explores this idea and proposes a visual programming language for usage in service-oriented architectures (chapter 8). This two-layer solution with different languages for coordination and computation (or execution) exists in other areas of computer science. In [106]

Gelernter describes a programming model with two different pieces, the coordination (composition) and the computation model, and develops a language focusing on coordination, *Linda* [107]. In [108], a similar approach is used to model parallel computing, where traditional sequential languages were used to write processing units whose concurrent behaviour was later specified with a graphical notation.

In the opposite direction of some approaches for the use of service-oriented architectures in the industrial environment [71] [74], where devices and service are normally small granulated, this approach tries to explore all the advantages of proprietary development platforms in order to achieve easier acceptance. In this proposal, all services are therefore holons representing a high level of functionality, which explore all the features of modern systems. A robot controller, p. e., is nowadays a complex and very powerful machine, with many advanced features like multitasking, advanced human-machine interfaces, and integrated force control. The use of service-oriented architectures from the ground up would imply full commitment from the robot manufacturer in order to provide services for all these features.

# 5 Service generation

---

In this chapter, the mechanisms for the automatic generation of services are approached. Section 5.1 gives us an overview of contracts in SOA , in section 5.2, industrial robot programs are used to integrate industrial robots in a service-oriented environment, and grammars from speech recognition systems are used to generate services in section 5.3. There are two conclusion chapters, one referring to the robot languages (section 5.2.5) and the other to the voice recognition grammars (section 5.3.5).

## 5.1 Contracts in SOA

Contracts are a key component in interoperability and are a legacy that SOA have inherited from component-based programming.

There are two ways of specifying services in a service-oriented environment: **contract-first design** and **code-first design**.

During the development of a consumer application, it is by far more popular to use a contract-based approach, since the contract already exists and clients want to follow it strictly. Considering the specification of servers (service providers), both code-first and contract-first approaches are commonly used. In Listing 8, a Web Services Description Language (WSDL) sample file is presented as a reference to evaluate the structure of a typical service contract file.

```

<?xml version="1.0" encoding="utf-8" ?>
<wsdl:definitions name="IdentificationService" targetNamespace="http://tempuri.org/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
...
xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex">
  <wsdl:types>
    <xsd:schema targetNamespace="http://tempuri.org/Imports">
      <xs:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://tempuri.org/">
        <xs:element name="Validate">
          <xs:complexType>
            <xs:sequence>
              <xs:element minOccurs="0" name="ID" type="xs:double" />
              <xs:element minOccurs="0" name="pass" type="xs:double" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="ValidateResponse">
          <xs:complexType>
            <xs:sequence>
              <xs:element minOccurs="0" name="ValidateResult" type="xs:boolean" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:schema>
      <xsd:import schemaLocation="http://localhost:8731/Service1/?xsd=xsd1"
namespace="http://schemas.microsoft.com/2003/10/Serialization/" />
    </wsdl:types>
    <wsdl:message name="IDService_Validate_InputMessage">
      <wsdl:part name="parameters" element="tns:Validate" />
    </wsdl:message>
    <wsdl:message name="IDService_Validate_OutputMessage">
      <wsdl:part name="parameters" element="tns:ValidateResponse" />
    </wsdl:message>
    ...<wsdl:message>
      <OtherMessages>
    </wsdl:message>
    <wsdl:portType name="IDServicePT">
      <wsdl:operation name="Validate">
        <wsdl:input wsaw:Action="http://tempuri.org/IDService/Validate"
message="tns:IDService_Validate_InputMessage" />
        <wsdl:output wsaw:Action="http://tempuri.org/IDService/ValidateResponse"
message="tns:IDService_Validate_OutputMessage" />
      </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="BasicHttpBinding_IDService" type="tns:IDService">
      ...
    </wsdl:binding>
    <wsdl:service name="IDService">
      <wsdl:port name="BasicHttpBinding_IDService" binding="tns:BasicHttpBinding_IDService">
        <soap:address location="http://localhost:8731/Service1/" />
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>

```

### Listing 8.WSDL sample code for a simple identification service

The code snippet presented in Listing 8 is the specification of a simple math Web Service using the *Windows Communication Foundation* (WCF) framework. This framework provides automatic tools to generate the WSDL file presented, starting from the C# class presented in

Listing 9. It is clear that the use of C# attributes is a powerful way of specifying services starting from an object-oriented language.

```
[ServiceContract]
public class IdentificationService
{
    [OperationContract]
    public bool Validate(double ID,double pass)
    {
        //validate user
        return true;
    }
    public double InternalOperation()
    {
        return "this operation is not present in the contract";
    }
}
```

**Listing 9. C# class that implements a simple identification service**

The contract-first approach is sometimes pointed out as the best approach to structure an application, leading the programmer to focus on interfaces, but since it is much harder to implement and WSDL is certainly verbose and largely non-intuitive to the beginner, the code-first approach has been used more often.

Sequential Function Charts is an example of a language where simplicity has shown to be more important than the capability of expressing complex systems logic. As such, it is our opinion that in an SOA industrial robotic cell, the specification of services should be made through a simple code-first mechanism.

## 5.2 Industrial robot programs

Industrial robots are still extensively programmed manually, nowadays (via the teach pendant). Therefore, the importance of robot programming languages is high, both from the industry's point of view, as a distinctive feature, and from the robot programmers', since it affects their productivity directly. In this section, robot programming languages are proposed as a way to specify contracts.

### 5.2.1 Robot programming languages

Since programming a robot is one of the key concepts that support flexible manufacturing, it is tempting to think that a robot program could be used in a code-first approach to the specifying of services within an SOA environment.

Each industrial robot manufacturer has designed its own language to program robots. Despite many efforts to unify robot programming languages (see p.e. [109]), there is no expectation for a single language in the future. All these languages are very different and have some unique characteristics. Considering five of the major manufacturers, ABB, Kuka, Motoman, Comau and Fanuc, we can see **three different concepts** for their programming languages. Motoman's programming language, INFORM III [110], is a high-level programming language, strictly close to robotics with no separation between data and functions (or procedures) that consume data, no support for structures, nor basic data types like boolean or string. Fanuc robots can be programmed with two different languages: Teach Pendant Programming [111], a high-level script-based programming language with the same limitations pointed out in INFORM III, and Karel [112], a pascal-like programming language. It is important to notice that, due to the existence of TPP, Karel is not very common in industrial installations. ABB's Rapid programming language [93], Kuka's Kuka Robot Language (KRL) [113] and Comau's PDL2 [114] are similar to Fanuc's Karel and fully support modularization, basic data types, structured/user-defined data types, and clearly separate data from functions. From this brief overview, it is clear that some robot programming languages are more suitable than others for the specification of services.

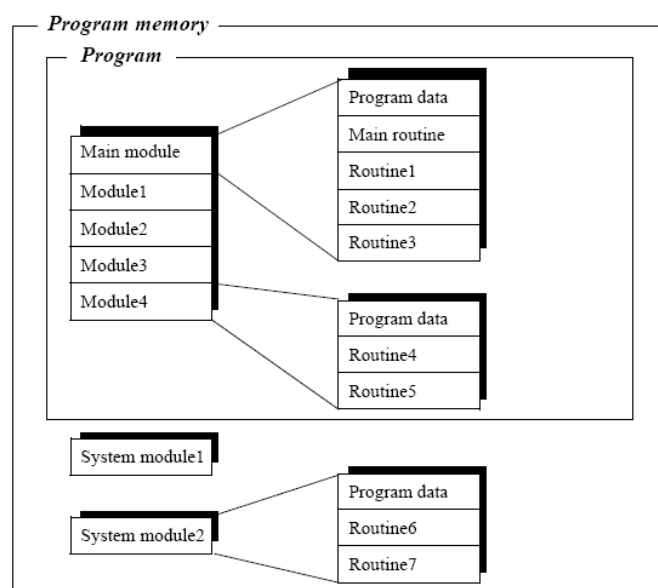
In this work, we will use ABB's Rapid language as the source specification for our services.

### 5.2.2 ABB's Rapid language

A Rapid program is divided into program modules and system modules, which are modules that are loaded whenever the system starts (Figure 24). There is a unique main module, which contains a global (public) entry procedure. There are three types of routines in Rapid: *Procedures*, *Functions* and *Traps*. Procedures and functions are equal except for the fact that procedures do not return a value. Trap routines deal with interrupts and cannot be called



directly. There are two different kinds of data types: atomic and record. Atomic types are not defined upon any other type and cannot be divided into parts or components. The internal structure (implementation) of an atomic type is hidden. The built-in atomic types are the numeric type num, the logical type bool and the text type string. Record data types are composed types. A record data type is a composite type with named, ordered components. The value of a record type is the composite value consisting of its components' values. A component can have an atomic or record type.



**Figure 24. ABB's Rapid program structure (Adapted from [93])**

It is worth noting that there are three kinds of data: **variable data**, whose value changes during program execution, **constant data**, whose value can only be assigned during programming, and **persistent data**, whose value is saved with the program working as a permanent memory of the program data. Rapid language supports the definition of user-defined structures through the reserved keyword RECORD.

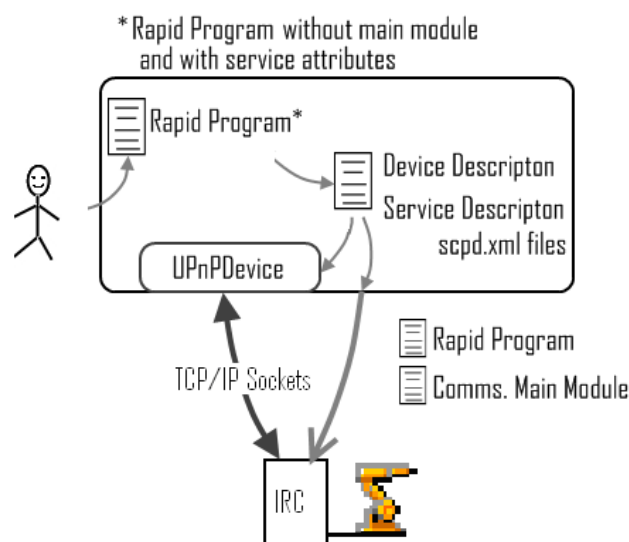
There are four different types of parameters: a **normal parameter**<sup>1</sup> serves as an input routine variable, meaning that any change to its value does not affect the corresponding argument; an

<sup>1</sup> The term "normal" is used throughout the text, led by the absence of a better term and following the robot manufacturer's example [93][115]

**INOUT parameter** specifies that a corresponding argument must be a variable or a persistent, which can be changed by the routine; a **VAR parameter** specifies that a corresponding argument must be a variable which can be changed by the routine; a **PERS parameter** specifies that a corresponding argument must be a persistent which can be changed by the routine.

### 5.2.3 Software developed

Since the robot controller does not support the development of network communications inside, the need arose for the use of an intermediate layer, which consists of a PC connected to the controller via a TCP/IP socket connection. The general software architecture is presented in Figure 25. In this figure, the grey arrows represent configuration stages and the black arrows represent communication on operation.



**Figure 25. Software Architecture**

During configuration, the user of the system, typically the system's integrator, writes the wanted rapid program and defines which part of it should be available in the network. This Rapid program is parsed and, as a result, a device description is generated. This device description is expressed in an xml format and points to SCPD files.

From the service description, two software applications are generated: a **rapid program** that handles the communication with the PC, and a **PC application** that works as a hub for all the communications to and from the robot. Both these applications are strictly related.

## 5.2.4 Extensions to Rapid language

In order to add the necessary expression power to the robot programming language, the choice has fallen over the use of **metadata annotations**. Metadata annotations provide a powerful way of extending the capabilities of a programming language and its runtime. Also known as **attributes**, these annotations can be directives that request the runtime to perform certain additional tasks, provide extra information about an item or extend the abilities of a type. Metadata annotations are common in a number of programming environments, including Microsoft's COM and the Linux kernel.

To match the basic elements of a service-oriented architecture in the device level, i.e., service, action and state variable, three attributes were chosen: the “![Service]” attribute can be used in a Rapid module and matches one UPnP service; the “![StateVariable]” attribute can be used in a Rapid variable and matches a UPnP StateVariable; and the “![Action]” attribute targets Rapid Procedures or Functions and matches UPnP actions. These attributes are used as Rapid comments, which allows the programmer to develop and test the program directly in the robot controller without changes. In Listing 10, a Rapid program with the attributes defined in this work is presented.

The “![Service]” attribute allows the robot programmer to organize the functionality of different robot programs present in the robot. Metadata annotations are usually placed before the language element and the same applies to the specification proposed here, with the exception of the “![Service]” element. Nevertheless, the attribute being inside a comment, the robot controller complains about its presence outside modules, and this element should therefore be placed after the name of the module (see Listing 10).

```
MODULE ABBIRC5Picker![Service]

PERS robtargt cam11:= [[253.4,-335.65,707.85],...];
PERS robtargt camAux := [[550,-499.41,463.17],...];
PERS robtargt box := [[18.40,-420.16,737.85],...];
```

```

PERS bool never_end:=FALSE;
PERS num progid:=30305;
VAR num auxX:=0;
VAR num auxY:=0;
VAR num auxIndex:=0;
VAR num auxIndexWhite:=0;
VAR string positionStr;
VAR bool okBool;

! [StateVariable]
PERS bool FinishedPick;

! [Action]
PROC PickAll(string positions)
    auxIndex := StrFind(positions,1,"#");
    WHILE (auxIndex<StrLen(positions)) DO
        positionStr:=StrPart(positions,auxIndex+1,StrFind(positions,auxIndex+1,"#")-auxIndex-1);
        auxIndexWhite:=StrFind(positionStr,1," ");
        okBool:=StrToVal(StrPart(positionStr,1,auxIndexWhite),auxX);
    okBool:=StrToVal(StrPart(positionStr,auxIndexWhite+1,StrLen(positionStr)-...));
        camAux:=cam11;
        camAux.trans.X:=auxX;
        camAux.trans.Y:=auxY;
        MoveJ cam11, v150, fine, tool0;
        MoveJ camAux, v150, fine, tool0;
        MoveL Offs(camAux,0,0,-46), v150, fine, tool0;
        Set DO06;
        Reset DO07;
        WaitTime 1.5;
        MoveL camAux, v150, fine, tool0;
        MoveJ Offs(cam11,0,0,30), v150, fine, tool0;
        MoveJ box, v150, fine, tool0;
        Reset DO06;
        Set DO07;
        WaitTime 0.8;
        Reset DO07;
        auxIndex:=StrFind(positions,auxIndex+1,"#");
    ENDWHILE
    IF ( FinishedPick=true) THEN
        FinishedPick:=false;
    ELSE
        FinishedPick:=true;
    ENDIF
ENDPROC
! [Action]
PROC Pick(num numToPick,string positions)
    ...
ENDPROC
ENDMODULE

```

**Listing 10. Sample robot program with service attributes**

The “! [Action]” attribute specifies that the annotated *Rapid* routine will have a UPnP counterpart. There is a restriction in routine parameters to the use of *Rapid* atomic types: bool, num and string, that will be matched to the *UPnP* types boolean, real (r8) and string, respectively. Only the *normal* type of *Rapid* parameters (see section 5.2.2 for the definition of *Rapid normal* type of parameters) is allowed. The other kinds of data, *INOUT*, *PERS* and *VAR*, imply an update of the value whenever it is changed inside the routine. The tempting solution

of using *UPnP out* arguments to tunnel the value update all the way until the UPnP layer would have led to a situation where two different values (a Rapid variable and an UPnP variable) would have to be refreshed and thereafter always synchronized. This would be achievable through a mandatory requirement that each bidirectional parameter should also be an *UPnP* state variable and that only those could be used as parameters. This solution would have been hard to explain to a user, and clumsy to implement, and the real added value is limited. Therefore, the option for a restriction to the **normal type** of parameters was made.

The *[StateVariable]* metadata annotation is used in every Rapid variable that is supposed to be available in the network. The restriction to atomic types is also applicable here, and only persistent data is allowed. This limitation is due to the technical implementation of the rapid TCP/IP socket server that is detailed in later paragraphs.

Rapid Service Generator (Figure 26) is the software application that materializes the concepts presented in this chapter. The **three main tasks** of this software application are: **parsing the Rapid program and building the device/service descriptions**; hosting an UPnP device that works like a **layer to expose the robot application in the service-oriented environment**; and finally, causing the **robot to communicate with the second layer provided by the application**.

The graphical human-machine interface allows the user to make some simple editing of the Rapid program, check if the program complies with the defined syntax, test UPnP actions with direct calls, download the program to the robot and start/stop UPnP service hosting.

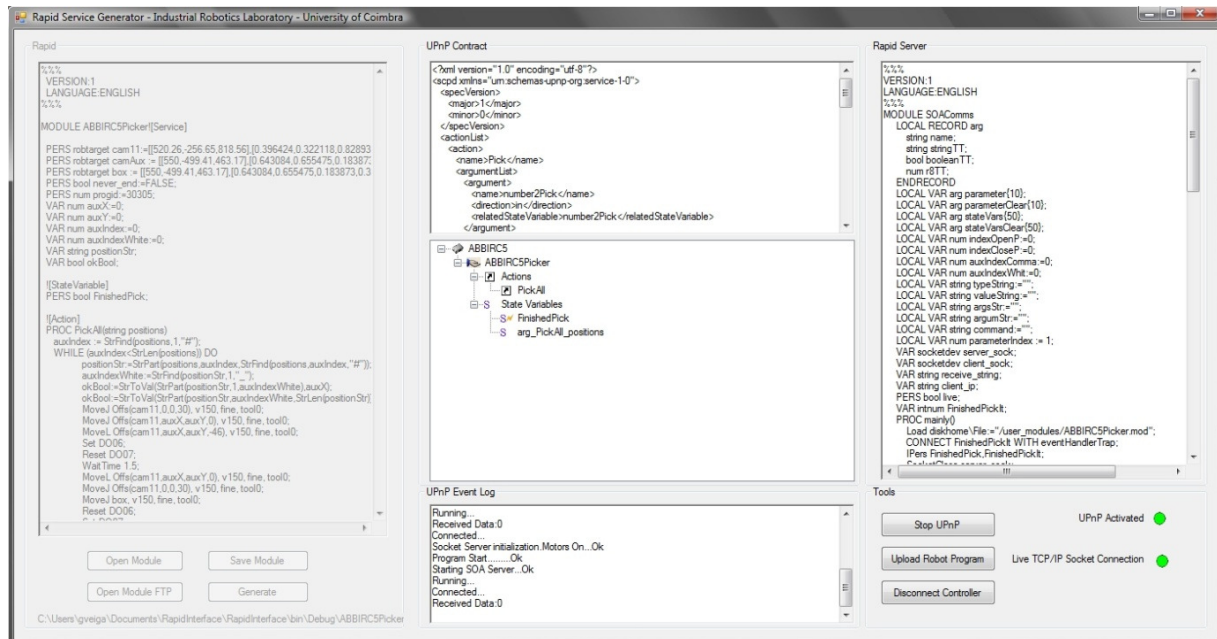


Figure 26. Graphical interface of the Rapid service generator

The *Rapid* TCP/IP socket server is composed by two major parts. The main procedure (Listing 11) is the one that will be running in the controller and is composed by: loading of the programmed module, creating socket connections, connecting state variables with *Trap* event handlers and a *while* structure that is waiting for call connections from the PC application.

```

MODULE SOACComms
  LOCAL RECORD arg
    string name;
    string stringTT;
    bool booleanTT;
    num r8TT;
  ENDRECORD
  LOCAL VAR arg parameter{10};
  LOCAL VAR arg parameterClear{10};
  LOCAL VAR arg stateVars{50};
  LOCAL VAR arg stateVarsClear{50};
  LOCAL VAR num indexOpenP:=0;
  LOCAL VAR num indexCloseP:=0;
  LOCAL VAR num auxIndexComma:=0;
  LOCAL VAR num auxIndexWhit:=0;
  LOCAL VAR string typeString:="";
  LOCAL VAR string valueString:="";
  LOCAL VAR string argsStr:="";
  LOCAL VAR string argumStr:="";
  LOCAL VAR string command:="";
  LOCAL VAR num parameterIndex := 1;
  VAR socketdev server_sock;
  VAR socketdev client_sock;
  VAR string receive_string;
  VAR string client_ip;
  PERS bool live;
  VAR intnum FinishedPickIt;
  PROC mainly()

```

```

Load diskhome\File:="/user_modules/ABBIRC5Picker.mod";
CONNECT FinishedPickIt WITH eventHandlerTrap;
IPers FinishedPick,FinishedPickIt;
SocketClose server_sock;
SocketClose client_sock;
SocketCreate server_sock;
SocketBind server_sock, "172.16.0.89", 2008;
SocketListen server_sock;
SocketAccept server_sock, client_sock \ClientAddress:=client_ip\Time:=120;
live:=TRUE;
WHILE live DO
    receive_string:="";
    SocketReceive client_sock \Str := receive_string\Time:=1;
    processArgs receive_string,command,parameter;
    TEST command
    CASE "PickAll":
        PickAll parameter{1}.stringTT;
    CASE "Pick":
        Pick parameter{1}.r8TT, parameter{2}.stringTT;
    ENDTST
ENDWHILE
ERROR
    IF (ERRNO=ERR_SOCK_TIMEOUT) THEN
        TRYNEXT;
    ENDIF
    IF (ERRNO=ERR_SOCK_CLOSED) THEN
        live:=FALSE;
        SocketClose server_sock;
        SocketClose client_sock;
        IDelete FinishedPickIt;
        TRYNEXT;
    ENDIF
    IF (ERRNO=ERR_LOADED) THEN
        UnLoad diskhome\File:="/user_modules/ABBIRC5Picker.mod";
        RETRY;
    ENDIF
ENDPROC

```

### Listing 11. Generated Socket Server - Part A

The use of Rapid *Trap* handlers to monitor the value of state variables is the reason for the limitation to the use of this kind of variables, since only these can be connected to this type of handlers [115].

The second part of the code implements the *Trap* routine that fires sockets to the PC application whenever a monitored variable changes, as described in Listing 12 where it is also possible to see the code that handles the internal messaging format that was defined for the robot UPnP server communications.

```

PROC processArgs(string in, INOUT string command, INOUT arg parameter{*})
    indexOpenP:=StrFind(in,1,"(");
    indexCloseP:=StrFind(in,1,")");
    command:="";
    parameter:=parameterClear;
    command:=StrPart(in,1,indexOpenP-1);
    IF (indexOpenP<=StrLen(in) AND indexCloseP<=StrLen(in)) THEN
        command:=StrPart(in,1,indexOpenP-1);
        argsStr:=StrPart(in,indexOpenP+1,indexCloseP-indexOpenP-1);
    ENDIF
ENDPROC

```

```

TPWRITE command;
TPWRITE argsStr;
auxIndex := 0;
typeString := "";
valueString := "";
parameterIndex := 1;
WHILE (auxIndex<StrLen(argsStr)) DO
  auxIndexComma:=StrFind(argsStr,auxIndex+1,",");
  argumStr:=StrPart(argsStr,auxIndex+1,auxIndexComma-auxIndex-1);
  auxIndexWhit:=StrFind(argumStr,1," ");
  typeString :=StrPart(argumStr,1,auxIndexWhit-1);
  valueString :=StrPart(argumStr,auxIndexWhit+1,StrLen(argumStr)-auxIndexWhit);
  TEST typeString
    CASE "stringTT":
      parameter{parameterIndex}.stringTT:=valueString;
    CASE "booleanTT":
      okBool:=StrToVal(valueString,parameter{parameterIndex}.booleanTT);
    CASE "r8TT":
      okBool:=StrToVal(valueString, parameter{parameterIndex}.r8TT);
  ENDTEST
  auxIndex:=StrFind(argsStr,auxIndex+1,",");
  parameterIndex := parameterIndex+1;
ENDWHILE
ENDIF
ENDPROC
TRAP eventHandlerTrap
IF FinishedPickIt=INTNO THEN
  SocketSend client_sock \Str := "boolean FinishedPick "+ValToStr(FinishedPick);
  TPWRITE "State Variable Changed:FinishedPick "+ValToStr(FinishedPick);
  TPWRITE "Event Fired";
ENDIF
ENDTRAP
ENDMODULE

```

### Listing 12. Generated ABB Rapid TCP/IP socket server - Part B

Videos from the presented application can be seen in Video 2 and Video 3. (Appendix B).

An important consideration needs to be made regarding the support of Rapid *functions*. Since the early stages of the development of this application, Rapid *functions* have been supported by the generation of an UPnP method with return value when a *Rapid* function was found. However, they should be used very carefully, since the UPnP answer is ruled by a timeout that can be shorter than the speed of the robot in executing a task and proceeding with the network answer. In fact, this situation is almost always present in the scenario proposed in chapter 4. In this approach, the idea of exploring all the capabilities of the robot programming language indicates that the exposed services should materialize into high-level functionality, which most certainly means that the robot will move. The movement of the robot implies a time delay that is incompatible with the UPnP timeout, which means that specifying Rapid *functions* as UPnP Actions is almost useless. The most flexible way to achieve coordination is to use a *Rapid* variable (properly networked with the corresponding metadata annotation) that can be



addressed at the end of the procedure, as shown in Listing 10, where a *Bool* variable, *FinishedPicking*, is used to notify the end of the pick operation.

## 5.2.5 Conclusions and future work

This work leaves interesting perspectives for a contract-based integration of a robot in a service-oriented environment. One crucial advantage of this approach is the complete compatibility with the existing robot technology. This means that current robot programmers can start integrating their robot applications in the network with no need for significant learning.

Experiments have been carried out with several users from three different groups: robot programmers (working in industry), engineering students, and advanced users (engineers from system integrators). The results from these experiments were very positive and the few corrections made had to do with graphical aspects of the interface.

A missing feature is the possibility to define data contracts. This is not a standard feature in UPnP middlewares, but it is a step forward that every SOA middleware is addressing. It is worth saying that the definition of data contracts should be considered in future industrial middleware implementations, since their success is highly dependent on standardization efforts. It is interesting to note that none of the users who evaluated the system seemed to mind the absence of advanced data support, which proves that, on a short-term basis, this solution addresses many issues in a flexible and simple way.

## 5.3 Speech recognition

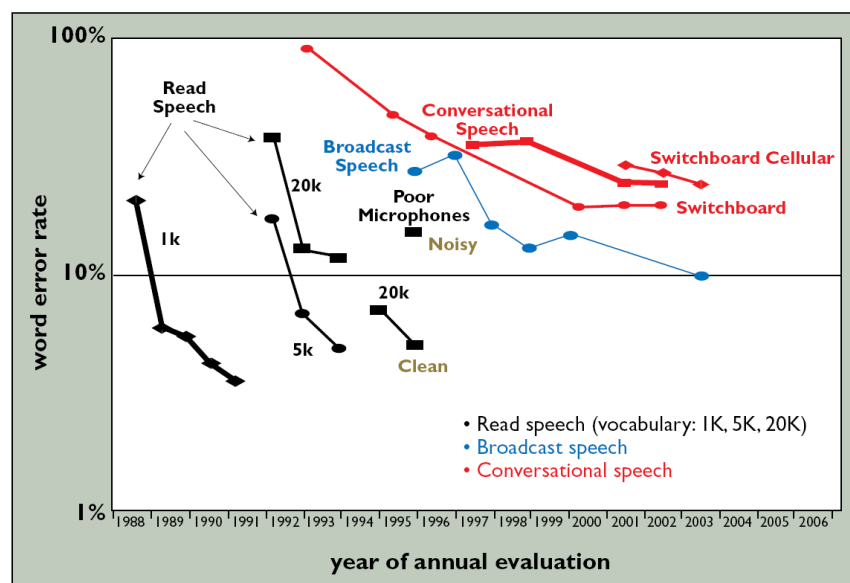
In the first Automatic Speech Recognition Systems (ASR)-related studies, grammatical and syntactical rules were used to validate speech. These approaches failed due to the numerous exceptions that human languages have, like dialects and accents, but also because when someone speaks, they do not enunciate each word separately. An important corner stone in the development of continuous speech systems (in opposition to former isolated word systems)

was the introduction of the **language models**, of which the n-gram [116] is by far the most used. These models predict the relation of a word with other n-words, statistically, and are very effective in guiding a word search for speech recognition, as shown by Claude Shannon [117] in a game where a computer defeated a human in guessing the next word in an arbitrary word sequence, whenever the span of words exceeds the number of 3.

During the 1980's and 1990's, great developments were achieved, especially through the refining of the statistical methods, namely the Hidden Markov Models (HMM).

These successes led the *Defense Advanced Research Projects Agency* (DARPA) to establish a program called EARS (Effective, Affordable, Reusable Speech-to-Text) [118][119], which promoted the development and widespread adoption of these technologies. The DARPA EARS program contains a series of tests designed to evaluate the performance of state-of-the-art recognition systems, which go from read speech with small vocabularies to recorded conversations from unacquainted adults (switchboard corpus).

Based on the data provided by this program, Deng [120] compiled data that clearly defines the evolution of the ASR capabilities (Figure 27).



**Figure 27. Historical progress of word error rates from speaker independent speech recognition for increasingly difficult speech data (Adapted from [120])**

Nowadays, the biggest challenges of voice recognition systems are related with **conversational speech**, where the user can speak to the machine as naturally as if they are talking to another human being. Considering data from [121], a human can recognize speech from 10 to 100 times more accurately than any machine, and it seems now clear that these problems cannot be solved without a strategy closer to the human way of processing speech. This statement includes a diversity of research vectors: phonetics, phonology and other principles of speech and language science [122]; integration of artificial intelligence in the HMM; bottom-up analysis instead of data-driven top-down [123][124]; semantic knowledge [125]; microphone hardware [126] and ergonomics [127].

Despite the need for further work on the recognition of conversational speech, **speech recognition technologies regarding read speech** are mature and ready to complement other means of human-machine interaction, which makes the business around speech recognition interesting for major computer industry players. Analysing the data from Figure 27, one can state that by 2003, the tests for broadcast speech (TV or radio) had already reached a level of significant quality (under 10% word error) and the read speech tests had been below this barrier for more than 8 years. Another interesting fact that can be extracted from Figure 27 is the influence of microphone quality in the results of the speech system (see 1995).

From these results, and particularly the ones concerning read speech, the business industry extracted different business models that are now part of a wide variety of products: mobile phones, GPS navigation systems, desktop PC software control and dictation software, among others. It is worth noting that, despite the quality of the ASR's that support these systems, their acceptance and real use are still small.

These products use two different modes that are usually supported by modern speech recognition systems: the dictation and the command modes.

In the **dictation mode**, data can be directly entered into the computer by speaking, allowing the user to compose an email or write a text. In this mode, the ASR tries to match the input with a complete language (English, for example).

In the **command mode**, the programmer of the system introduces a grammar (a new layer over the global grammar) that limits the amount of possible commands. In this way, considering a well-defined grammar, these systems provide better accuracy and performance, and reduce the processing overhead required by the application.

Even though the support for the dictation mode has been available in modern operating systems and commercial software packages since 2002 (Windows XP or Dragon NaturallySpeaking) with good levels of performance, their acceptance is still limited outside some niche markets like healthcare, for instance [128]. There are many possible explanations for this, and they range from the social acceptance of “speaking to a machine” to the use of poor microphones that degrade the performance in a way that makes the system useless.

From a **practical point of view**, the most robust (speaker-independent) voice-enabled systems present in the market **are still working in command mode**. The grammars used in these systems are explained and evaluated in section 5.3.3.

### 5.3.1 Speech platforms

Although the challenge of speech systems is mainly centred in recognition, there is also a set of complementary technologies that promote their integration in broader multimodal interaction systems. Within these complementary technologies, there are four that play a major role: **voice synthesis, dialogue management, Web Integration** and **Call Control (for phone applications)**. To address all these technologies in an integrated way, the W3C consortium has proposed a standard, VoiceXML [129], which embraces them all uniformly:

- SRGS – Speech Recognition Grammar Specification[130];
- SIRS – Semantic Interpretation for speech recognition [131];
- SSML – Speech Synthesis Markup Language[132];

- CCXML - Call Control eXtensible Markup Language[133];
- SCXML – State Chart eXtensible Markup Language[134] (expected to be the dialog management language of VoiceXML);

In this work, major attention is paid to speech recognition and interpretation, due to their greater relevance to industrial systems, as described in section 5.3.2.

### 5.3.2 Speech for Industrial automation

The industrial use of voice-enabled systems is a promising concept. Consider for example a line operator who has both hands occupied, one holding a polishing tool and the other holding the product. The ability to control the conveyor, to set the spindle speed or even some extra equipment with the voice provides substantial added value. In the same way, a MIG-MAG welder can tune welding parameters in the middle of a seam, without needing to stop the procedure.

The idea of using voice recognition in industrial systems is not new (see [135] [136]), but the full exploitation of this concept suffered from the applicability problems of general ASR systems, further enhanced by the characteristics of the industrial environment, which raises some extra challenges concerning safety or the reliability of the recognition. Nowadays ASR systems, when used in command mode with restricted grammars, have finally given researchers the ability to develop systems with an industrial level of robustness [137], and have supported the development of the first voice recognition commercial solutions for industrial automation [138].

### 5.3.3 Speech recognition grammars

A **grammar defines the words and patterns of words** that a user can say at any particular point in a dialogue. When a programmer specifies a grammar, they define a set of words and patterns of words to be recognized by the system. These grammars have the expression power of a Context-Free Grammar (CFG)[139].

In Listing 13, a sample grammar (SRGS/SIRS format) is presented that describes a small set of commands to operate a desktop PC. Each rule is identified by an ID and it is possible to verify how rules can be made by composition, with optional elements and repetitions.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE grammar PUBLIC "-//W3C//DTD GRAMMAR 1.0//EN"
    "http://www.w3.org/TR/speech-grammar/grammar.dtd">
<grammar xmlns="http://www.w3.org/2001/06/grammar" xml:lang="en"
    ...
    version="1.0" mode="voice" root="basicCmd">

  <rule id="basicCmd" scope="public">
    <example> please move the window </example>
    <ruleref uri="#command"/>
  </rule>
  <rule id="command" >
    <ruleref uri="#action"/>
    <ruleref uri="#object"/>
  </rule>
  <rule id="action">
    <one-of>
      <item weight="10"> open   <tag>1</tag> </item>
      <item weight="2">  close <tag>2</tag> </item>
    </one-of>
  </rule>
  <rule id="object">
    <item repeat="0-1">
      <one-of>
        <item> the </item>
        <item> a </item>
      </one-of>
    </item>
    <one-of>
      <item> window </item>
      <item> file </item>
      <item> menu </item>
    </one-of>
  </rule>
</grammar>
```

**Listing 13. Sample SRGS grammar.**

When a speech recognizer matches an audio input with a grammar rule, it can produce a literal text transcription of the detected input. A recognizer may be capable of, but is not required to, perform subsequent processing of the raw text to produce a **semantic interpretation** of the input. In Listing 13 some tags were added to one of the rules (action) that associates an integer number with the recognition of the open or closed actions. In this grammar format, it is also permitted to process the output via the use of *ECMAScript* [140].

In spite of the standard presented above, a group of different grammar formats has been used over the last few years, which possess a similar expression power:

- W3C Speech Recognition Grammar Specification (SRGS)[130]

- Microsoft's SAPI 5 XML.[141]
- Java Speech Grammar Format – (JSFG) [142]
- GSL (Nuance's Grammar Specification language)[143]

It is important to notice a general movement towards the W3C standard, with its adoption by Nuance, Microsoft (with the purchase of another company, Tellme) and Sun (the W3C standard is actually based on the JSFG).

### 5.3.4 Voice recognition integration in robotic work-cells

Recent SRS can be used in the industrial environment, see [137]. This type of technology can be integrated seamlessly in an SOA middleware because it is extensively event-driven. To achieve this integration, a software application that allows the automatic pairing of voice recognition events with UPNP events has been developed (Figure 28).

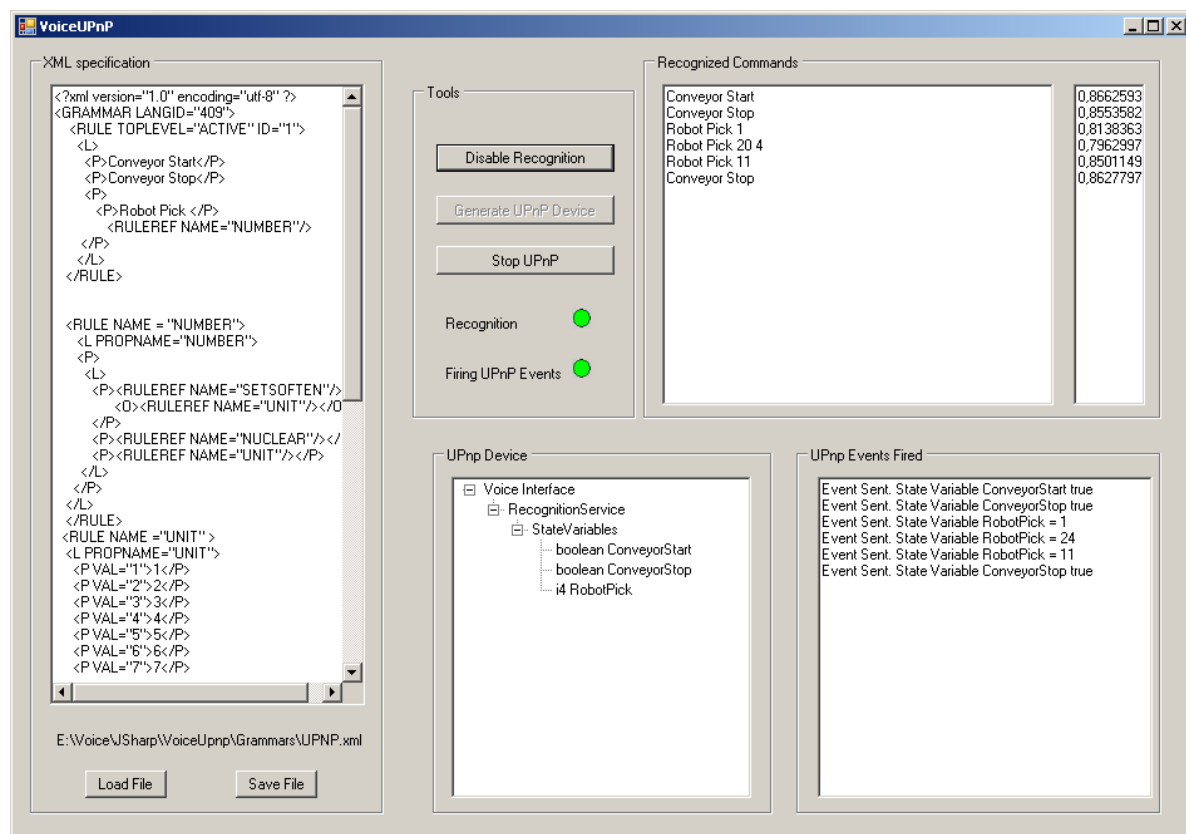


Figure 28. Voice Recognition Interface (SAPI 5.1)

The SRS selected for use within this work was the Microsoft speech engine included in the *Microsoft Speech Application Protocol Interface* MSAPI 5.1 [144]. This system includes an *automatic speech recognition* (ASR) engine and a *text to speech* (TTS) engine.

To create an UPnP device that can publish events corresponding to ASR events, an application was developed that implemented the following strategy: first, the XML grammar was parsed and an XML-DOM (Document Object Model) tree document created; this tree was afterwards traversed and UPnP state variables dynamically created and added to the `RecognitionService` of the voice interface device. All combinations implemented with grammar tags `<L><\L>` (List) were listed, and a Boolean state variable created for each one of them. The name of the state variable was the recognized sentence without spaces. Nevertheless, if this traversal method went through each rule reference, a very high number of variables would be created. To avoid these difficulties and to express the real mean of the recognized number, an integer state variable was associated with each of the recognitions that could contain a number. It is important to notice that the UPnP events were fired every time a new value was assigned to the state variable, even if the value was the same.

Grammars are used to define what the ASR should recognize. Each time a sequence defined in the grammar is recognized, an event is fired by the SRS. The Microsoft SAPI allows three different ways for specifying grammars: included in the code (programmatic grammars), using XML files, or using CFG files. Since XML is a well-accepted standard, it has been used to specify speech recognition grammars.

Grammars define a `TopLevel` Rule that include all the necessary commands. From each of these commands it is possible to call other rules. In the example presented in Figure 28, a rule (“NUMBER”) was created to support the recognition of numbers (0-99). This rule is composed by several secondary rules (UNIT, SETSOFTEN,...) that have associated properties.

These properties allow the easy recovery of a value when a number is recognized, because they are sent as an argument of the delegate call when a recognition event occurs (Listing 14).

```
public void handleRecognition(int StreamNumber, System.Object StreamPosition,  
                             SpeechRecognitionType RecognitionType,
```



```

        ISpeechRecoResult Result)
{
    SpeechDisplayAttributes a = Result.PhraseInfo.GetDisplayAttributes(0, 1, false);
    SpeechEngineConfidence confidence = Result.PhraseInfo.Rule.Confidence;
    confidence.ToString();

    double num = 0;
    string cmd = "";
    if (confidence != SpeechEngineConfidence.SECLowConfidence)
    {
        txtBoxRecoCmd.Text += Result.PhraseInfo.GetText(0, -1, true);
        if (Result.PhraseInfo.Properties != null)
        {
            if (Result.PhraseInfo.Properties.Count > 0)
            {
                foreach (ISpeechPhraseProperty p in Result.PhraseInfo.Properties)
                {
                    if (p.Name == "CMD")
                    {
                        cmd = (string)p.Value;
                    }
                    if (p.Name == "UNIT"
                        || p.Name == "SETSOFTEEN"
                        || p.Name == "NUCLEAR")
                    {
                        int lixo = (int)p.Value;
                        num += (double)lixo;
                    }
                }
            }
            if (num != 0)
            {
                if (this.btn_Advertise.Text == "Stop UPnP")
                {
                    device.GetService("RecognitionService").SetStateVariable(cmd, num);
                    this.txtBoxUpnpEvents.Text +=
                        "Event Sent. State Variable " + cmd + " = " + num;
                }
            }
            else
            {
                Handling of non-numeric variables
            }
            txtBoxUpnpEvents.Text += System.Environment.NewLine;
        }
    }
    string curr;
    curr = Result.PhraseInfo.GetText(0, -1, true);
    int aux = confidence.ToString().IndexOf("Confidence");
    this.txtBoxConfidence.Text += confidence.ToString().Substring(3, aux - 3);
    txtBoxConfidence.Text += System.Environment.NewLine;
    this.txtBoxRecoCmd.Text += System.Environment.NewLine;
}
}

```

#### Listing 14. Recognition handling delegate: retrieving semantic properties

This application provides a very interesting approach to linking the meaning of both dictated numbers and UPnP state variables. This approach could be extended to terms like Conveyor and Robot, which could be associated with their respective devices, or even linked to ontology in robotics.

The application presented in section 5.3.4 has been further developed in order to integrate SRGS grammars with SIRS semantic interpretation. In this new application, the semantic analysis is done via *JavaScript* and further processed in code.

### 5.3.5 Conclusions

The importance of the automatic generation of services is shown again in these applications. With the XML momentum on the IT technologies, the number of declarative programming languages opens interesting perspectives for the techniques presented here. A good example are the recent tools for the development of GUI presented by major software players, Microsoft eXtensible Application Markup Language (XAML) [145] or Sun JavaFx [146]. These languages declaratively describe the user interface and rely on object oriented languages (Java or C#) for handling events generated by the user interaction, avoiding in this way the use of partial classes and other techniques to cope with the large amount of code present in graphical interface libraries. An exploratory test has been made with XAML files, with the description of a graphical interface being linked to UPnP network events:, in a similar way as the software presented here, and the results are promising.

*Things should be made as simple as possible,  
but not simpler*

*Albert Einstein*

# 6 Robotic work cell programming

---

In the context of the solution presented in chapter 4, the integration of devices in a robotic work-cell is basically a two-step process: first services are generated relying on device specifications, and afterwards they are integrated making use of orchestration techniques with graphical support. This chapter describes the **proposed orchestration language for device-level SOA networks**.

In the following sections, the **composition of internet level services** is revised (section 6.1), the **device-level orchestration techniques discussed** (section 6.3), and finally the **composition language proposed** for the SME robotic cell environment (6.5). Furthermore, and due to the graphic nature of the proposed language, a brief overview on **graphical programming languages** is made in section 6.2

## 6.1 WS\*- Web Services Composition

As presented in section 2.4.1, the goal of WS\* - *Web services* is to exploit XML technologies in order to promote cross-machine applications that can be published, located, and invoked over the Web. Besides the mechanisms presented in section 2.4.1 for the description of interfaces (WSDL [56]), the discovery (UDDI [55]) and the invocation of services (SOAP [57]), the

integration of distributed applications calls for the combination of services that in this text, according to [147], will be called service composition. The concept behind the composition of services is derived from the one introduced by component-oriented programming, which, like in many other WS\*-related technologies, was extended and standardized using XML specifications. The relevance of the composition to E-Business has promoted a spate of different proposals from major players: *XLang* from Microsoft [148], *Web Services Flow Language* from IBM [149], *Web Services Choreography Language* from HP [150], all promoting different interaction patterns among services. These patterns have led to the coining of several terms: choreography, orchestration, automation, coordination, collaboration, and conversation, which characterize the composition of services. From the evolution process, a standard has emerged as the most promising: *Business Process Execution Language for Web Services* (BPEL4WS) [151].

Web service composition is extensively revised in [152], with special emphasis on the composition's main issues: coordination, transaction, context, conversation modelling and execution modelling.

### 6.1.1 Business Process Execution Language for Web Services

With the purpose of getting a clear picture of the *BPEL4WS* language, a small example is presented in the following paragraphs. Bearing in mind the simple validation service presented in Listing 8, let us define a BPEL process called *ScheduleLaboratory*. The initiation of this service will be triggered by a request from a consumer that needs an ID validation. Among other operations, the BPEL4WS process will in turn use the validation functionality provided by the simple Identification service. The WSDL specification presented in Listing 8 defines an operation in the port *IdentificationService*, *Validate*, and their respective messages: *IDService\_Validate\_InputMessage*, *IDService\_Validate\_Output\_Message*. BPEL4WS compositions rely deeply on these WSDL specifications, which are completed with the definition of two extra messages, *ScheduleLaboratoryMessage* and *BalanceManager*, and an extra port to handle the requests coming from the consumer (Listing 15).

```
<wsdl:message name="ScheduleLabMsg">
```

```

    <wsdl:part name="user" element="xsd:string" />
    <wsdl:part name="hourBegin" element="xsd:double" />
    <wsdl:part name="hourEnd" element="xsd:double" />
    <wsdl:part name="password" element="xsd:string" />
  </wsdl:message>

  <wsdl:message name="SchedulingResultMsg">
    <wsdl:part name="schedulingOk" element="xsd:bool" />
  </wsdl:message>

  <portType name="scheduleLabPT">
    <operation name="scheduleLab">
      <input message="ScheduleLabMsg"/>
      <output message="SchedulingResultMsg"/>
    </operation>
  </portType>

```

**Listing 15. Extra portType and messages required for the consumer input**

To standardize the relationship between services and consumers, the BPEL4WS specification has defined the concept of partner. The XML tag *plnk* is the one used to define partners, and defines up to two roles that refer to the *portTypes* that are provided and required by any two services it links together. The relation can be either unilateral or bilateral, depending on whether the service requires one or more ports. In the presented example (Listing 16), this *partnerLinkType* will be used to define the relation between the customer and the process, as well as between the process and the *IdentificationService*. Only one role is required because neither the process nor the *IdentificationService* requires their respective consumers to support another *portType*.

```

<plnk:partnerLinkType name="schedulingPLT">
  <plnk:role name="scheduleService">
    < plnk:portType name="scheduleLabPT"/>
  </plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType name="validatingPLT">
  <plnk:role name="identificationService">
    < plnk:portType name=" IDServicePT"/>
  </plnk:role>
</plnk:partnerLinkType>

```

**Listing 16. Partner link and variable definition for BPEL4WS**

After these type definitions, the missing part is the definition of the process itself Listing 17. Due to its executable nature, a BPEL4WS process needs to create instances both for variables (XML tag *variables*) and for partners (XML tag *partners*). In this example, two containers are defined: one holds the data provided by the consumer, and the other the result that comes from the validation process. A process can contain one or more activities of different types. In

Listing 17, an activity of the type *sequence* is defined, that can be started through the `<receive>` activity once this has the *createInstance* attribute set to *true*. After this step, the process asks the WebService created before to validate the identity of the user. This is done through a regular Web service invocation, defined in the process by an `<invoke>` activity that uses the message in its input container as an argument, and puts the answer it gets into its output container.

```
<process name="loanApprovalProcess"
  targetNamespace="http://robotics.dem.uc.pt/addbillProcessing"
  xmlns="http://schemas.xmlsoap.org/ws/2002/07/business-process/"
  <partners>
    <partner name="customer"
      partnerLinkType="schedulingPLT"
      myRole="scheduleService"/>
    <partner name="validator"
      partnerLinkType="validatingPLT"
      partnerRole="identificationService"/>
  </partners>

  <variables>
    <variable name="request" messageType="ScheduleLabMsg"/>
    <variable name="approvalInfo" messageType="SchedulingResultMsg"/>
  </variables>

  <sequence>
    <receive name="receive1" partner="customer"
      portType="scheduleLabPT"
      operation="scheduleLab" container="request"
      createInstance="yes">
    </receive>
    <invoke name="invokevalidation"
      partner="validator"
      portType="scheduleLabPT"
      operation="scheduleLab"
      inputContainer="request"
      outputContainer="approvalInfo">
    </invoke>
    <reply name="reply" partner="customer" portType="scheduleLabPT"
      operation="scheduleLab" container="approvalInfo">
    </reply>
  </sequence>
</process>
```

#### Listing 17. BPEL4WS sequence definition

The last *reply* activity is used by the process to answer the customer's request. The *portType* of the reply should be equal to the one of the receiver, in order for the process to figure out who to send the reply to. After the reply, the process ends and the execution is terminated.

It should be noticed that, for the sake of simplicity, this small example skips some details like namespaces and xml headers, and also misses important features of the BPEL4WS, like p.e.

fault handling, complex activities like the ones related with control flow, security, bi-lateral partner links and scopes, among others.

### 6.1.2 Semantic web services and ontology

BPEL4WS, introduced in the last section, is mainly known as a manual composition model. From an internet programmer's perspective, WebServices are seen as the door for machine-to-machine communications. It may have been due to this fact that the developments in the research for automatic service composition appeared even before the widespread adoption of WebServices, in a time where little to no manual composition existed [103][153][154]. The most frequently used definition of ontology in the literature on semantic web services [155][156] is that an ontology is a collection of web services that share the same domain of interest, and describe how web services can be described and accessed. There are several ontology languages, RDF (*Resource Description Framework*), DAML+OIL, or OWL, and most of them target the web services market, namely DAML-S (DARPA Agent Markup Language for Web services) and OWL-S, that define formal specifications. These ontologies can therefore be queried and provide the means to define sophisticated class properties. It should be noticed that all these efforts popped up after the year 2000, specifying a proprietary notation to extend WSDL, and leading to a specification mess. W3C made an effort, completed in 2007, to standardize the semantic notations in WSDL: SAWSDL (*Semantic Annotations for Web Services Description Language*).

## 6.2 Visual Programming Languages in industrial automation

Visual programming languages have been known since 1969 and applied to every type of domain. Their main use in modern systems is related to domain-specific languages [157]. One of the main fields of application for visual programming is industrial automation. In the history of PLC programming, *Sequential Function Charts* and *Function Blocks* are among the most

successful programming languages from the IEC-61131<sup>1</sup> specification [102]. Indeed, this success has influenced the development of the modern standard IEC-61499, that is itself a vpl. The graphical development environment Labview [158] is a very well established software tool for data acquisition, instrument control and industrial automation, which relies on a graphical data-flow programming language, sometimes called *G*. Robotics is another area where graphical programming seems to have found some space with the introduction by one manufacturer of an icon-based programming environment, *Morpha* from *Kuka Roboter* [159].

### 6.3 Analysis - Device-Level service composition

In the device-level, the problem of the orchestration specification was at the beginning of this thesis mainly unchallenged. Within the European project SIRENA (the first EP that arose using device-level SOA for industrial automation), the orchestration subject was approached [160], but only under the perspective of reusing business level languages. In SIRENA's child projects, SOCRADES and SODA, several similar efforts can be found, see p.e.[161]. These research efforts passed through the manual orchestration directly to problems related with automatic reasoning systems (automatic orchestration), without questioning the adequacy of the existing orchestration languages. In fact, this problem is mentioned in a SODA report [162], where several different possibilities for orchestration languages are questioned: process-based, event-driven and data-driven approaches. Referring to the **process-based approach**, inherited from BPEL4WS, this report refers that this **language needs enhancements for dynamic discovery of services and the use of events**.

The SME scenario that is the target of this thesis can be classified as a reactive system (see section 2.5), i.e. mainly event-driven, which is one of the reasons for the success of device-level SOA dealing with this type of interactions. Therefore, the orchestration language/technique needs to make a rupture with BPEL4WS.

---

<sup>1</sup> It can be found in old literature as IEC-1131, according to IEC's old numbering scheme.



Furthermore, from the evaluation made in chapter 3, one interesting detail became prevalent: although the graphical capabilities of the MVPL were a major advantage for most of the users during the learning process, the more concrete **event/action** structure presented by the simple programmer interface (Figure 17) revealed easier to understand. Furthermore, some users pointed out the resemblance of *Sequential Function Charts* as an advantage in the comprehension process, which is mainly due to the reactive nature of the industrial processes.

For the reasons described earlier, i.e, the absence of a standard for the needs of device-level orchestration, a new orchestration language is proposed based on **statechart formalism**.

## 6.4 StateCharts

The *Statechart* formalism was described by David Harel [163] in 1987 as facilitating the design of complex discrete-event systems. *Harel Statecharts*, or simply *Statecharts*, is a visual formalism that extends the finite-state automata formalism (traditional state-transition diagrams) with the notions of hierarchy, concurrency, history, and communication. Statecharts are hierarchical, i.e., a state may contain another statechart down to an arbitrary depth, which highlights the modularity and clustering of the visual formalism. These characteristics combined with the Or decomposition, which is **the mutual exclusion among a group of states**, are the fundamentals of the abstraction capabilities that Harel [163] said were missing from flat statemachines. As a result, **state levels** can be created within statecharts, and the designer can hide complexity whenever zooming out of the state chart without losing detailed resolution when zooming in is required. Two or more statecharts may be run in parallel, which means that their parent state is in two states at the same time. This permits **state orthogonality**, which means independence and concurrency, and is achieved via the **And decomposition**. Another important feature of statecharts is the possibility to **hold historical information** inside a state, conditioning the re-entrance into that state.

### 6.4.1 Statecharts eXtensible Markup Language

Statechart XML (SCXML) is a W3C specification [134] that can be described as an attempt to render Harel Statecharts in XML. The aim of this standard is to provide a basis for future standards in the area of multimodal dialogue systems. Even though this effort is being carried out by the W3C group for voice technologies, SCXML provides a generic state-machine based execution environment and a modern (XML) state machine notation for control abstraction. In fact, SCXML is a candidate for control language within multiple markup languages coming out of the W3C.

Consider for example the microwave oven model presented in Figure 29.

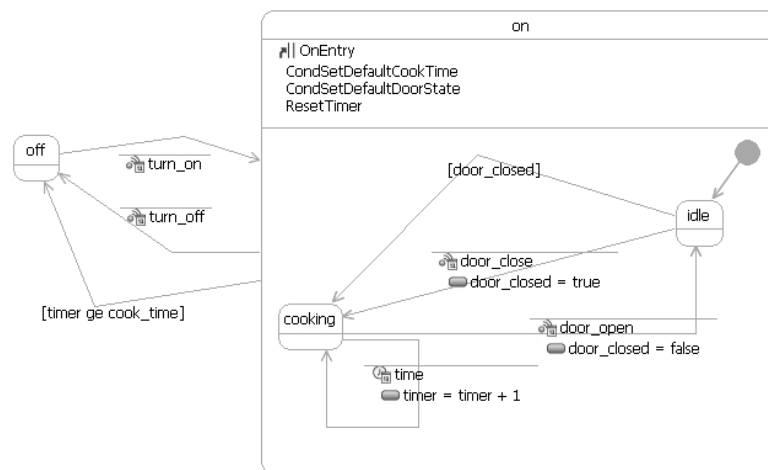


Figure 29. Microwave oven

The equivalent SCXML specification is:

```
<?xml version="1.0"?>
<scxml xmlns=
  "http://www.w3.org/2005/07/scxml"
  version="1.0"
  initialstate="off">

  <state id="off">
    <!-- off state -->
    <transition event="turn_on">
      <target next="on"/>
    </transition>
  </state>
  <state id="on">
    <initial>
      <transition>
        <target next="idle"/>
      </transition>
    </initial>
  </state>
  <state id="idle">
    <!-- idle state -->
    <transition event="door_close">
      <target next="cooking"/>
    </transition>
    <transition event="door_open">
      <target next="off"/>
    </transition>
  </state>
  <state id="cooking">
    <!-- cooking state -->
    <transition event="time">
      <target next="cooking"/>
    </transition>
  </state>
</scxml>
```

```

        </transition>
    </initial>
    <onentry>
        ...
    </onentry>
    <transition event="turn_off">
        <target next="off"/>
    </transition>
    <transition cond="${timer ge cook_time}">
        <target next="off"/>
    </transition>
    <state id="idle">
        <transition cond="${door_closed}">
            <target next="cooking"/>
        </transition>
        <transition event="door_close">
            <assign name="door_closed" expr="${true}"/>
            <target next="cooking"/>
        </transition>
    </state>
    <state id="cooking">
        ...
    </state>
</state>
</scxml>

```

### Listing 18. SCXML sample specification

As it can be seen in this example, an SCXML statechart can be divided into two major parts: the first one is composed by the machine states and their corresponding transitions, and the other by the executable content.

The SCXML executable content consists of actions that are performed as part of making transitions and entering and leaving states. The executable content is responsible for the modification of the data model, for raising events and invoking functionality on the underlying platform. It is worth noting that executable content cannot cause a state change, or fire a transition, except indirectly, by raising events that are then caught by transitions. This separation in the specification leaves room for platforms to add executable content corresponding to special features.

## 6.5 Proposed language and tests

The language proposed for the orchestration of the work-cell was designed with simplicity in mind. The evaluation made with the users revealed that the orchestration language should be easily understandable and should be a thin layer, excluding many features, like mathematical,

for instance. It is our opinion that for complex calculations the use of general programming languages like C# or Java is still a must, but that is not the target of this work; SME's are.

An example of this language is presented in Listing 19. The language is very similar to the SCXML except for the direct use of network events and actions. In this way, the need to write additional code, as with normal SCXML integrations, is avoided.

```
<?xml version="1.0"?>
<stateMachine xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" namespace="RoboticsLab" name="CellController"
initialState="Stopped" stateMachineType="Active">
  <state id="Stopped" historyType="None">
    <transition event="urn:schemas-upnp-org:device:VoiceInterface:1;RecognitionService;ConveyorStart" target="Conveyor Running">
      <UPnPAction name="urn:schemas-upnp-org:device:Conveyor:1;GeneralMove;InitAuto" />
    </transition>
  </state>
  <state id="Conveyor Running" historyType="None">
    <transition event="urn:schemas-upnp-org:device:Conveyor:1;GeneralMove;SensorCamera" target="Getting Pieces">
      <UPnPAction name="urn:schemas-upnp-org:device:SmartCamera:1;whiteobjectdetection;getPos" />
    </transition>
  </state>
  <state id="Getting Pieces" historyType="None">
    <transition event="urn:schemas-upnp-org:device:VoiceInterface:1;RecognitionService;RobotPick" target="Robot Picking">
      <UPnPAction name="urn:schemas-upnp-org:device:ABBIRC5Picker:1;ABBIRC5Picker;Pick">
        <UPnPArgument name="numToPick" val="urn:schemas-upnp-org:device:VoiceInterface:1;RecognitionService;RobotPick" />
        <UPnPArgument name="positions" val="urn:schemas-upnp-org:device:SmartCamera:1;whiteobjectdetection;ansGetPos" />
      </UPnPAction>
    </transition>
    <transition event="urn:schemas-upnp-org:device:PDAInterface:1;FormInteractionEvents;RobotPick" target="Robot Picking">
      <UPnPAction name="urn:schemas-upnp-org:device:ABBIRC5Picker:1;ABBIRC5Picker;Pick">
        <UPnPArgument name="numToPick" val="urn:schemas-upnp-org:device:VoiceInterface:1;RecognitionService;RobotPick" />
        <UPnPArgument name="positions" val="urn:schemas-upnp-org:device:SmartCamera:1;whiteobjectdetection;ansGetPos" />
      </UPnPAction>
    </transition>
  </state>
  <state id="Robot Picking" historyType="None">
    <transition event="urn:schemas-upnp-org:device:ABBIRC5Picker:1;ABBIRC5Picker;FinishedPick" target="Stopped">
      <UPnPAction name="urn:schemas-upnp-org:device:Conveyor:1;GeneralMove;ForceForward" />
    </transition>
  </state>
</stateMachine>
```

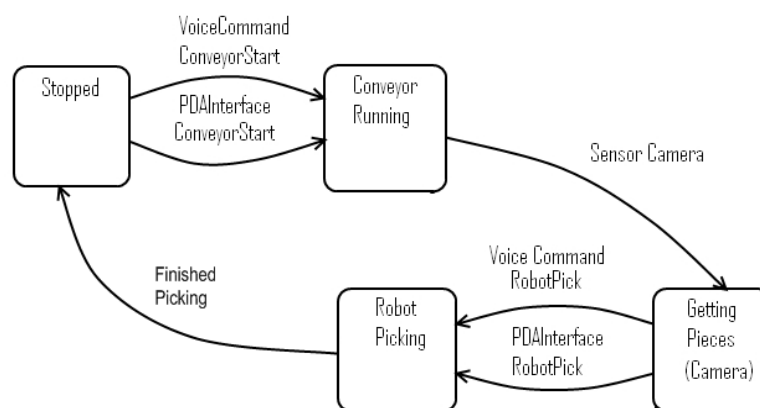
#### Listing 19. XML program for the conveyor cell

To demonstrate the concepts discussed earlier, the test-bed used in chapter 5 was extended with an alternative form of command, a PDA Interface. This application was developed with the programming stack developed during this work that was described in section 3.9.



**Figure 30. PDA Interface**

The PDA interface includes the same functionality as the Voice Command: **Conveyor Start**, **Conveyor Stop**, **Robot Pick [number of points]**. With these commands, the user of the work-cell is supposed to alternatively use the voice interface or the PDA. This orchestration (Figure 31) matches the orchestration program listed in Listing 19, and demonstrates the use of alternative state transitions, each of them with actions associated.



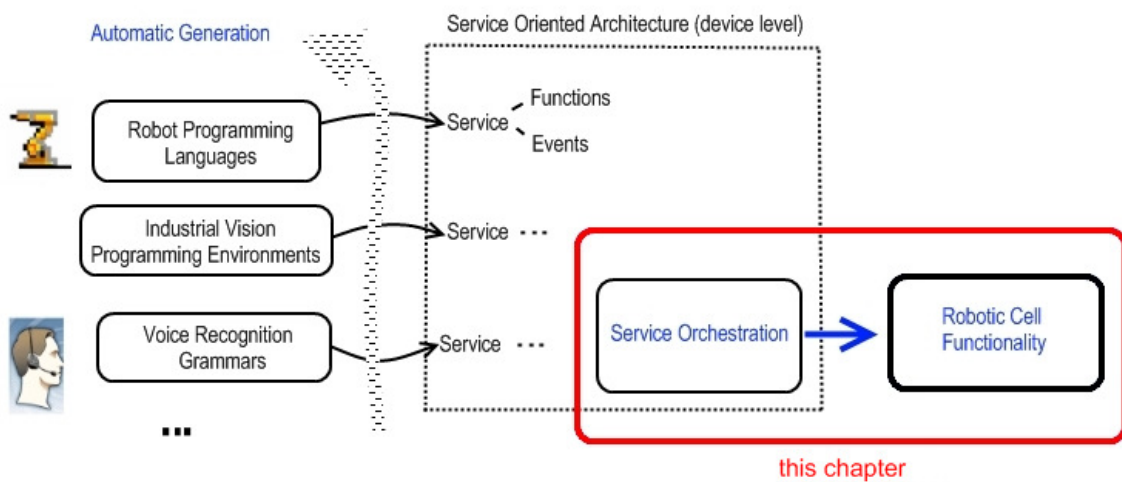
**Figure 31. Statechart with control via Voice or PDA**

In this small example, the transition from the interaction stages is triggered by events raised by two different devices: VoiceInterface or PDAInterface.

The language is similar to the SCXML specification, but the actions and events are directly routed to the network.

### 6.5.1 Software developed for service orchestration

The software described in this chapter materializes part of the solution presented in chapter 4, which handles the composition of services (red square in Figure 32).

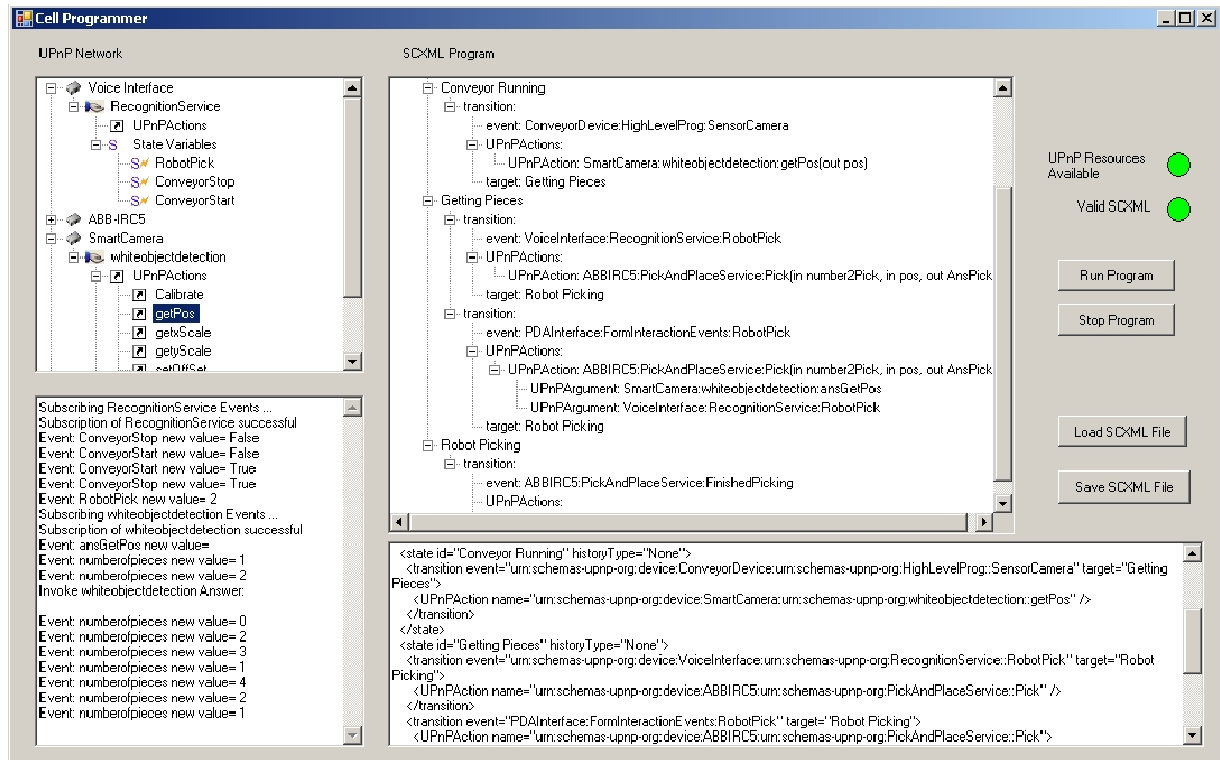


**Figure 32. General Architecture - Cell Programming**

The software developed can be divided into two distinct parts: the implementation of the **statechart engine**, and the **user interface itself**.

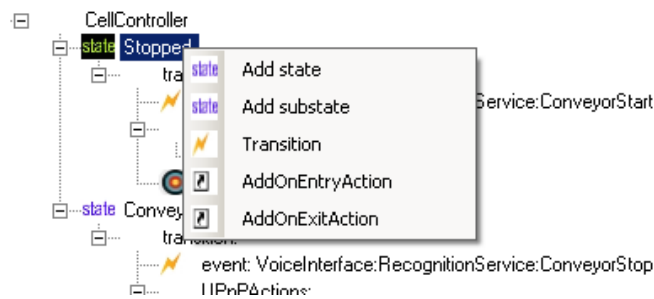
Nowadays there are too few SCXML implementations available, and the most notable effort is *CommonsSCXML* [164]. Since *CommonsSCXML* is still in an 0.x version, and the need arose to extend its standard functionality, it was decided in a first approach to develop an SCXML engine from scratch. The application presented in this paper was developed in C# following the guidelines presented by Miro Samek in [165], and extended with the basic part of the SCXML language. Considering the W3C standard (Barnett et al. 2007), the implementation presented here does not include the *Extensions to the basic State Machine Model* and the *Executable Content*.

The application developed includes two main parts: on the left side of the graphical application form there is an **UPnP generic control point** that sniffs the network, and a log for UPnP dealings, like UPnP events, discovery notifications and subscriptions; on the upper right side of the form, a tree view of the **statechart** is presented, and on the lower part, the respective SCXML file.



**Figure 33. Cell Programmer**

The system is basically programmed in two steps. The first stage is the composition of the state machine via a context menu over the *treeview* that represents the statechart (Figure 34).



**Figure 34. State machine composition context menu**

The second stage is fundamentally a drag’n’drop application between the service network and the state machine. UPnP actions can be dragged into state machine actions, either entry or exit, and UPnP events can be dragged to state machine events. The identification of the type of service uses the generic template mechanism described in 3.1.2 and concatenates device service and event in the following way:

```
urn:schemas-upnp-org:device:VoiceInterface:1::RecognitionService::ConveyorStart
```

**Listing 20. Conveyor start event specification**

The UPnP discovery mechanism described in 3.1.1 includes the definition of Hello/Bye messages as well as “keep alive” messages. Making use of these features, a validation of the program is performed in order to guarantee the presence of all the UPnP resources needed.

Two videos of the system working can be seen in Video 4 and 5 (Appendix B).

## 6.5.2 Evaluation and improved versions

Due to the specificities of the SME scenario related with usability and user-friendly features, the tests developed with the proposed software are of major importance. The example presented here was compared with an alternative solution developed in [91] that used TCP/IP sockets as the main protocol, and the reduction of orchestration programming time was of 40%. Comparing with the MVPL orchestration, the amount of time required was similar.

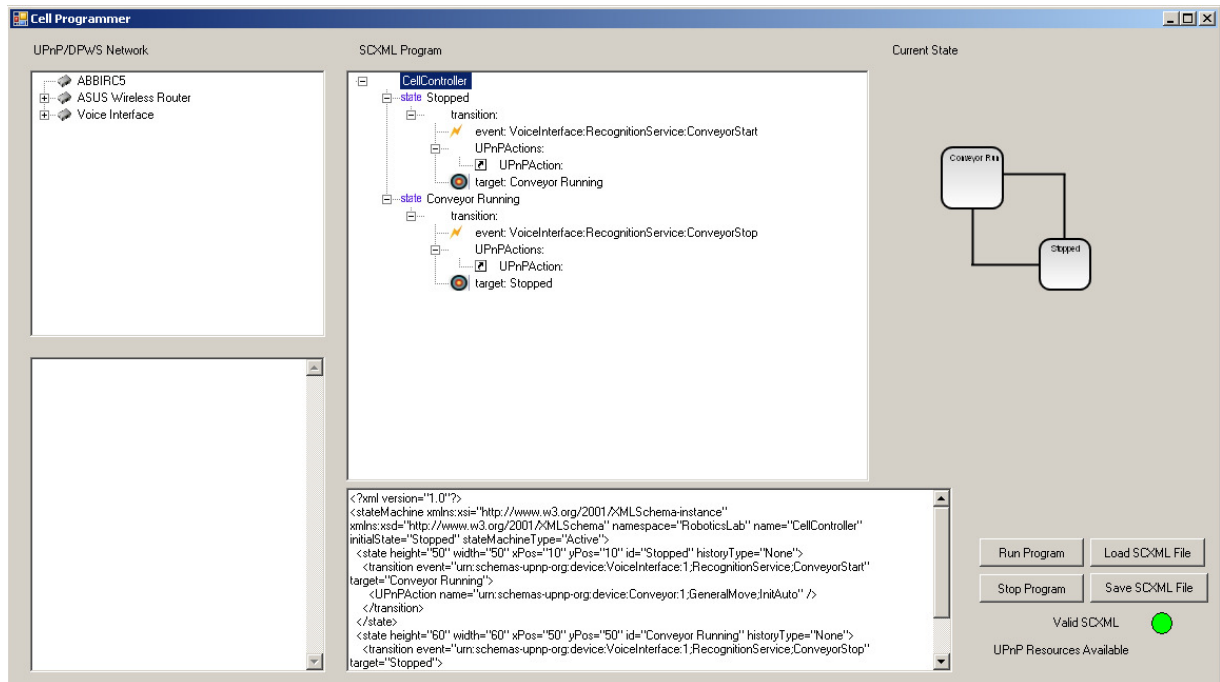
The definition of the orchestration language promotes a clear separation of concerns: orchestration and computation, with a clear definition for orchestration, which promotes



reusability and comprises a good basis for automatic reasoning systems. A clear example of this advantage is presented in section 7.4.

Another evaluation was made based on the feedback of a robot programmer, an R&D engineer from a system integrator, and an engineering student. The results are satisfactory and, comparing with the results obtained with the MVPL, the simplicity and the expressiveness of the statecharts, usually recalling SFCs, made them easier to learn. Another advantage noted was the separation of the cell logic from a specific device, creating a simpler way to store that information. Moreover, users claimed that the existence of a concrete cell program combined with the discovery features of the SOA platform will power a better reconfiguration experience. In such a system, the user would be able to select a given cell program, that in turn would “ask” the available services to verify if all the necessary services were available in a compact and standardized way. Good reviews were also given to the idea of integrating software application like the one presented in Figure 33.

This approach’s main drawback, as described by some users, was the lack of some programming features: simple math operations, loops, conditional statements. These problems, very common in visual programming languages, are usually solved with the correct service definition, but that was not always the case in the first interaction during the tests. Another problem had to do with missing graphics for the statecharts. To address this issue, the second version of the software, currently being developed, includes a hybrid program representation, with both treeview and graphical statechart.

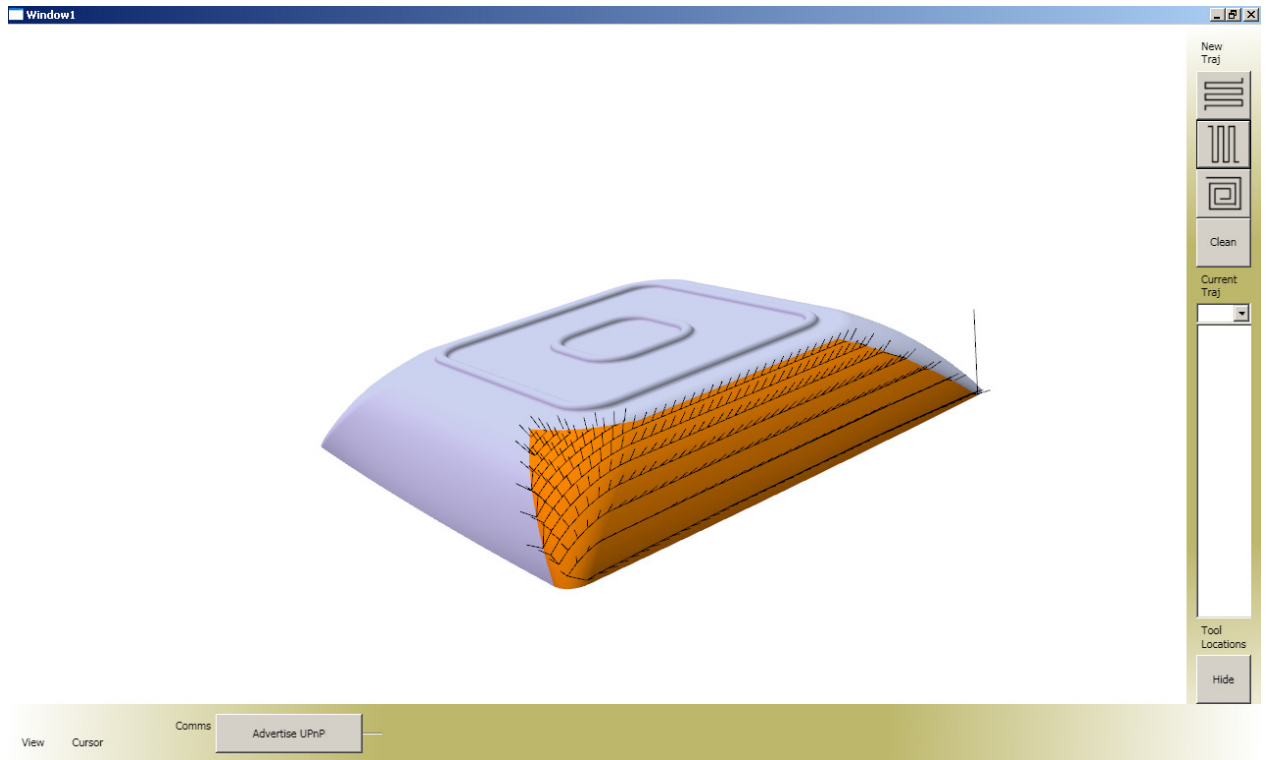


**Figure 35. Prototype new version of the Cell Programmer**

This new version includes several improvements to the statemachine engine, including more features like conditioned transitions, and the network support is being extended to the DPWS platform.

### 6.5.3 Other Tests

Several other tests, including many laboratory developments, have used this platform as an orchestration basis. They range from the integration of new HMI devices like the Wiimote (Video 6 Appendix B), to the development of a 3D graphical interface that allows the user to define simple trajectories over surfaces and then provides access to those trajectories over an UPnP (and later DPWS) service (Figure 36).



**Figure 36. 3D trajectory generator UPnP service.**



*I visualize a time where we will be to robots  
what dogs are to humans, and I'm rooting for the machine*

*Claude Shannon*

# 7 Automatic configuration

---

In this chapter, fast configuration for work-cells is pursued. First, a short overview on PbD work-cells is given due to the importance of these systems' configuration stage (section 7.1). Afterwards, a cell specification language is defined.

## 7.1 Configuration of PbD cells

**Programming by demonstration (PbD)** is a key enabler for the future of robotics in SMEs. With the increasing number of robot manufacturers with **low-level force control integrated** into their controllers, see [29] for example, and the **easy access to robust voice recognition engines**, the use of programming-by-demonstration systems can be generalized [31][166]. Furthermore, an increasing number of force-torque sensors will create a market that can make the sensor prices fall, either through reduced production cost or through the use of different technologies, thus raising the competitiveness of these systems. In a PbD system, the operator guides the robot by hand and uses speech to record positions, enabling I/O and waiting for I/O, among others, just as if s/he was **teaching another worker**.

In this type of scenario it is important to define names for the components, not only to disambiguate communication (as with the human operator) while teaching a new task, but also

to parameterize the ASR. The robustness of the recognition needed for industrial applications is obtained in **command mode** (see section 5.3), which uses **restricted grammars** that need to be written. One can claim that these grammars can also be taught using dictation mode, but that would imply the loss of robustness and an increased operator dependence.

Looking into the **setup process of a work-cell**, three major steps can be defined: installation, configuration, and programming. **Installation** includes the hardware installation, either mechanical or electrical. **Configuration** includes I/O naming, coordinate systems definition and network communications setup, among others. **Programming** includes all the process-related steps, which obviously include the robot programming.

To achieve a plug-n-produce experience, all three stages, setup, installation and configuration, need to be improved. However, looking into the PbD scenario described above, the importance of the **programming stage** is reduced due to the implicit reprogramability of such systems, and the relevance of the **configuration stage** enhanced due to the referred needs for grammar definition of the voice recognition system.

## 7.2 Cell functionality specification

Pursuing a leaner configuration stage, this work defines a cell description language that should be sufficient for the automatic configuration of a programming by demonstration system. There are currently several work-cell description languages, mainly to be used inside off-line programming systems [167][168] that include graphical information but also, in some cases, functionality. For SMEs, this type of software solution is not usually adequate due to high costs and the required access to CAD data of all equipments.

With this in mind, this work proposes the use of a cell functionality specification to be used in SMEs. An example of this specification is presented in Listing 21.

```
CellSpec Basic
{
  rob "Robot"{
    robtarg "Corner 2";
    robtarg "Hole";
  }
}
```

```

robtarget "Home";
do1 "air";
do2 "table lock";
ao1 "air pressure";
ao6 "maximum force";

tool "drill"{
    ao2 "speed";
    do3 "motor";
    do4 "motor direction";
};
routine "Drill hole";
integer "XAmount";
};
}

```

### Listing 21. Cell specification example

In the already described PbD scenario, the operator in charge of this work-cell can operate the robot and its peripherals using voice commands with concrete meaning for the application. Furthermore, he can define routines composing movements and I/O management.

One can question the added complexity brought by an extra specification, and the acceptance of this by SME users. This is a known issue that is not tackled here, but several recent developments in computer science have opened some possible solutions: natural processing languages and ontology-based reasoning systems.

From a simple point of view, natural language processing is the extraction of useful information from text written in an informal (natural) way. This is a promising research field, with some valuable results having recently been presented. In [169], these technologies are being used to generate 3D views of car accidents based on the information extracted from their protagonists' descriptions. Similarly, a cell specification like the one presented in Listing 21 can be obtained from the informal description provided by the SME user. On the other hand, the growing movement towards the introduction of semantic information weaved inside *Web Service* technologies will soon be replicated in Web Services made for automation equipment. This semantic information can later be matched against a robotics ontology, thus providing means for reasoning on functionality. The cell specification proposed here can be seen as an intermediate product in this process.

The use of this cell specification can be very wide. In Figure 37, a general view of possible uses for this specification is presented. The system's integrator (or maybe even the end user) will

describe the work-cell according to the defined format or, alternatively, an ontology-based automatic generation can be provided. Compiler technologies are afterwards used to generate the necessary execution code that will provide a faster installation/reconfiguration. In the depicted case, the information is used to generate a **voice recognition grammar** with the basic commands for operating the work-cell, the **robot Rapid code** that matches the voice commands, **documentation for the user**, a **robot configuration file** that defines the I/O names and, finally, an **SCXML** that orchestrates the voice commands and the robot routines.

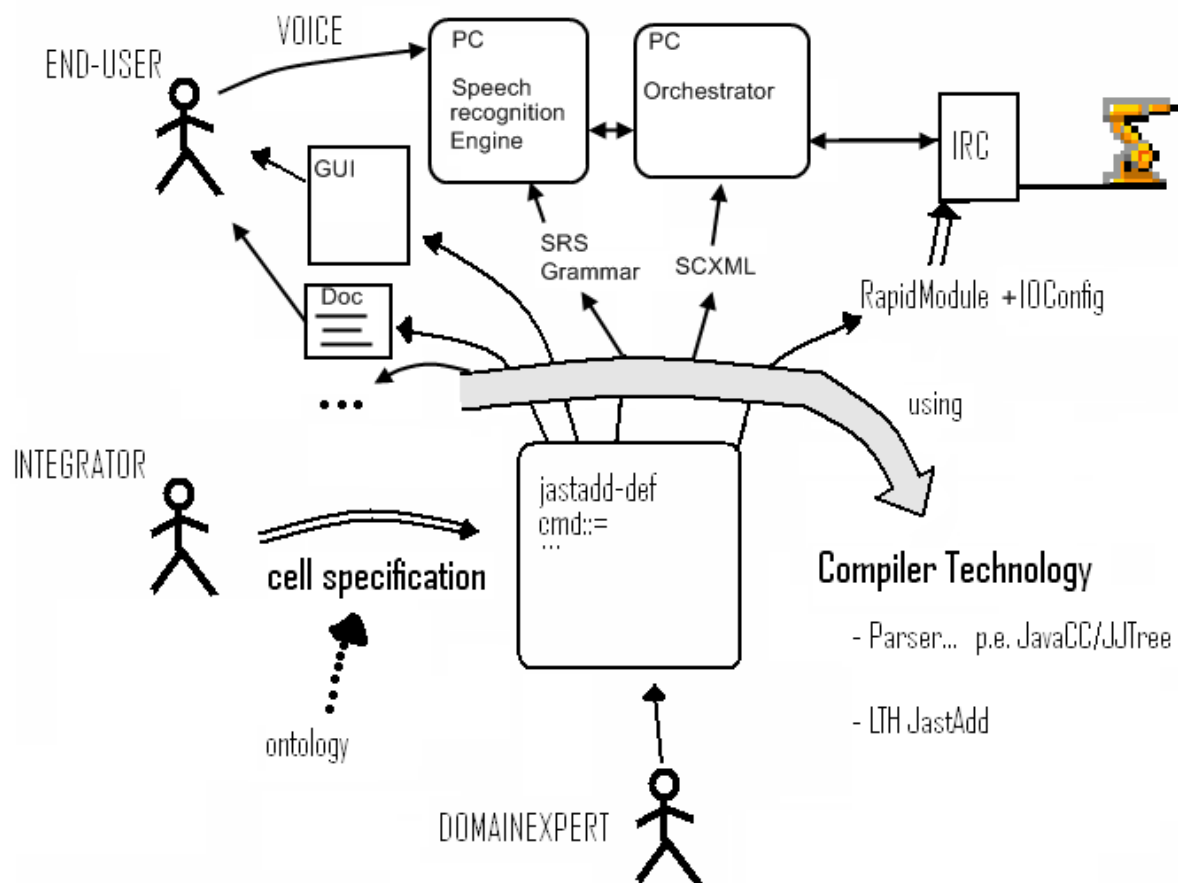


Figure 37. Automatic generation of speech grammar and robot code



## 7.3 Software developed

The software that demonstrates the concept was developed using the *JastAdd* [170], and is essentially composed by a *front-end* that deals with the cell-specification format, and four *back-ends* that target the referred documents.

Section 7.3.1 briefly introduces *JastAdd* using specifications from the actual software as examples.

### 7.3.1 JastAdd

JastAdd is a Java-Based compiler construction system that combines features including object orientation, aspect orientation, declarative features, reference attribute grammars and context dependent rewriting. The major goal of JastAdd is to allow high-level abstractions to promote extensibility and modularity in compiler construction. Nowadays, state-of-the art compilers do not support this type of fine-grained extensibility, which was largely shown through the execution of large language compilers in [171] with Java.

JastAdd works over abstract syntax trees and is therefore dependent on other tools for lexical analysis, parsing and construction of the said tree. In this work, *JavaCC* and *JJTree* [172] were used for the first two phases.

JavaCC is an LL(*k*) parser generator, short for *leftmost derivation, left-to-right parse with k symbol lookahead* (for example see [173]), which generates a parser for a formal grammar provided in EBNF (*Extended Backus–Naur Form*) notation. This formal grammar is given in terms of a *concrete syntax*, which describes the syntactic rules of the language. By contrast, the *abstract syntax* represents the internal structure of the program.

The basis for JastAdd specification is an abstract syntax tree that describes the programs of a language in terms of typed tree data structures, which is a common technique in compiler technologies. An *Abstract Syntax Tree* (AST) provides a good representation of the program, with each node referring roughly to one language element, and constitutes a good tool for analysis and synthesis steps. JastAdd uses an **object-oriented abstract grammar specification**,

from which a Java class hierarchy is generated. In this specification, a non-terminal is declared by its name, followed by the symbol `::=`, after which the children of the non-terminal are listed. A specification of a non-terminal is called a production, and matches a node that only appears on the left side of a production. Considering the grammar presented in Listing 22 as an example, the `CellSpec` is a non-terminal and the `<ID:String>` is a terminal. The nomenclature for the inheritance in the object-oriented sense is expressed through the use of a single semi colon (see `MoveCmd` that inherits from `Cmd`), and the definition of abstract elements is described with *abstract*. Multiple children from the same type compose a list that can be expressed with an asterisk.

```
CellSpec ::= SysList;
SysList ::= Sys*;
abstract Sys ::= Decl;
RobSys:Sys ::= DeclList /CommandSet:Command*/;
DeclList ::= Decl*;
Command ::= Use Cmd;
abstract Decl ::= IdDecl;
abstract Cmd;
abstract MoveType;
MoveCmd:Cmd ::= TypeReference;
SavePosCmd:Cmd ::= TypeReference;
SetDOCmd:Cmd ::= TypeReference;
SetAOCmd:Cmd ::= TypeReference;
SaveToRoutineCmd:Cmd ::= TypeReference;
SetIntegerCmd:Cmd ::= TypeReference;
UnitIncrIntegerCmd:Cmd ::= TypeReference;
JointJogCmd:Cmd ::= JointId Direction;
LinearJogCmd:Cmd ::= LinearAxeId Direction;
RobDecl:Decl ::=;
RobTargetDecl:Decl ::=;
DODDecl:Decl ::=;
AODDecl:Decl ::=;
RoutineDecl:Decl ::=;
IntegerDecl:Decl ::=;
ToolDecl:Decl ::= DeclList;
TypeReference ::=;
Use ::=;
LinearAxeId ::= <LDIGIT:String>;
JointId ::= <JDIGIT:String>;
Direction ::= <DIRECTIONID:String>;
IdDecl ::= <ID:String>;
IntExp ::= <INT:String>;
```

**Listing 22. Abstract grammar for cell specification**

When designing a compiler, a programmer usually needs to modularize some features in terms that are not related to language constructs, and are therefore hard to fit directly in the methods from the class hierarchy described before. To address these issues, compilers usually rely on modularization techniques that crosscut this class hierarchy, like the visitor pattern [174] or aspect-oriented programming. Jastadd uses the second (more powerful) technique

supporting a **static aspect-oriented programming** model, similar to the open classes presented by *AspectJ* [175]. These modules allow behaviour that crosscuts the AST class hierarchy to be specified in one single place, and the methods and fields are weaved in the classes later. Traditional examples of these aspects are name analysis, type analysis, error checking and code generation, that can be specified separately despite referring to the same classes. In addition to enhancing readability, this makes it possible to add or remove specific aspect modules during the implementation or debugging, and to reuse modules for different versions of the compiler. In the current application, crosscutting behaviour is clear in the specification of several aspects related to the generation of different outputs. Listing 23 shows one of these aspects, that allows the generation of the configuration file for an ABB robot, with the particularity of being related to only one part of the AST specification.

```

aspect EIOGen {
    public String CellSpec.EIOGen(String indent) throws IOException{
        StringBuffer buffer = new StringBuffer();
        buffer.append(getSysList().EIOGen(indent+"  "));
        return buffer.toString();
    }
    public String SysList.EIOGen(String indent){
        StringBuffer buffer = new StringBuffer();
        for(int i = 0; i < getNumSys(); i++){
            buffer.append(getSys(i).EIOGen(indent));
        }
        return buffer.toString();
    }
    abstract String Sys.EIOGen(String indent);
    public String RobSys.EIOGen(String indent) {
        StringBuffer buffer = new StringBuffer();
        Stream s = new FileStream(".\\XMLResult\\EIOHeader");
        StreamReader sr = new StreamReader(s, Encoding.UTF8);
        buffer.append(sr.ReadToEnd());
        for(int i = 0; i < getDeclList().getNumDecl(); i++){
            buffer.append(getDeclList().getDecl(i).EIOGen(indent));
        }
        return buffer.toString();
    }
    public String DODDecl.EIOGen(String indent) {
        StringBuffer buffer = new StringBuffer();
        buffer.append("-Name "+getIdDecl(). -Type \"DO\" -Unit \"IODIG\"");
        return buffer.toString();
    }
}
...
}

```

**Listing 23. Aspect for the code generation of the I/O robot configuration file**

JastAdd supports the declarative formalism of *Reference Attributed Grammars* (RAGs). Attributes are defined by equations that can be found in the node itself (*synthesized attributes*)

or in ancestor nodes (*inherited attributes*). This formalism extends traditional attribute grammars with the support of reference attributes, which in the compiler's case are references to nodes that can be anywhere in the AST. This is typically used to connect identifiers to their declarations. Listing 24 is a *jrag* module that defines an attribute *env* representing the environment of visible declarations. With this attribute, the computations over the AST regarding visibility became easier to implement.

```
aspect PrintingBindingsRAG {
  syn String Decl.name;
  syn String IdDecl.name()=getID();
  syn lazy DeclList IdUseList.decl()=env().lookupType(type());
  inh Sys Sys.env();
  inh Sys CommandSet.env();
  inh Sys Command.env();
  inh Sys DeclList.env();
  inh Sys Decl.env();
  inh Sys Cmd.env();
  inh Sys TypeReference.env();
  inh Sys MoveType.env();
  inh Sys Use.env();
  eq SysList.getSys(int index).env()=null;
  eq RobSys.getCommandSet().env()=this;
  eq RobSys.getDeclList().env()=this;
  eq CommandSet.getCommand(int index).env()=env();
  eq Command.getUse().env()=env();
}
```

**Listing 24. Jrag for binding declarations with their uses**

One powerful feature is the ability to enlarge the AST with nodes that are defined by equations, rather than generated by the parser. These nodes are called *Non-terminal attributes* [176] (NTAs) due to the fact that they are both similar to non-terminal nodes and attributes, and are typically used in cases where a child node cannot be constructed at parse time. NTAs are defined by equations and may be dependent on the structure of the AST. They provide a powerful mechanism, since they allow the AST to be modified and new nodes to be added during the execution of the compiler. The use of NTAs has been particularly suitable for the software developed, because the generation of voice commands implied extra knowledge that had not been originated during the parsing. Looking into the cell specification presented in Listing 21, one suitable voice command for the work-cell would be, p.e., “Robot Move Joint to Corner2”. The information in this command could not be directly obtained from parsing, and to provide good integration in the AST, it should be integrated via NTAs. The definition of these non-terminal attributes is made via a *Jrag* file (Listing 25). In this case, several voice commands

are linked to the robot system specification, and their details connected to specific type references.

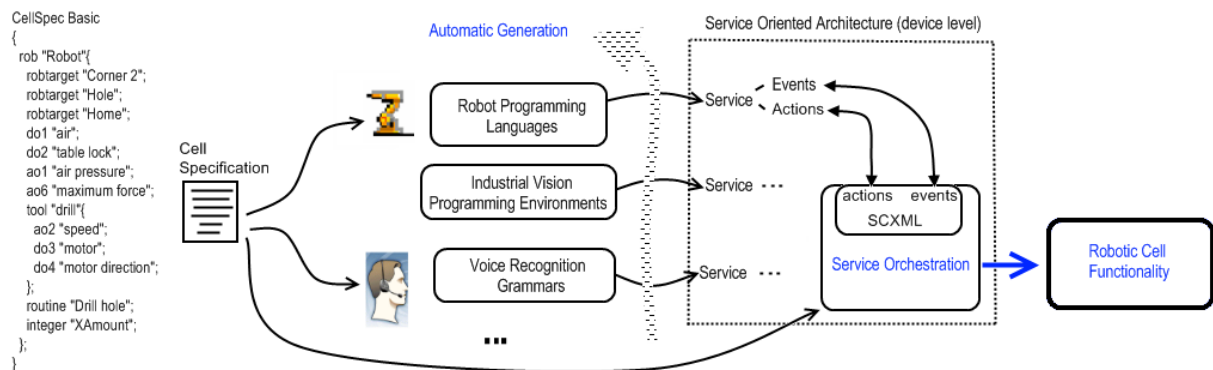
```
aspect CommandSet {
  // Define the NTA
  syn lazy List RobSys.getCommandSetList() {
    return new List()
      .add(new Command(
        new Use(), new SetDOCmd(new TypeReference()))))
      .add(new Command(
        new Use(), new SetAOCmd(new TypeReference()))))
      .add(new Command(
        new Use(), new MoveCmd(new TypeReference()))))
      .add(new Command(
        new Use(), new SavePosCmd(new TypeReference()))))
      .add(new Command(
        new Use(), new JointJogCmd(new JointId(), new Direction()))))
      .add(new Command(
        new Use(), new LinearJogCmd(new LinearAxeId(), new Direction()))))
      .add(new Command(
        new Use(), new SaveToRoutineCmd(new TypeReference()))))
      .add(new Command(
        new Use(), new SetIntegerCmd(new TypeReference()))))
      .add(new Command(
        new Use(), new UnitIncrIntegerCmd(new TypeReference())));
  }
}
```

**Listing 25. Jrag with non-terminal attributes.**

Moreover, *JastAdd* also supports *rewrites* [177], which enable the rewriting of the abstract syntax tree using contextual information, but they were not used in this work.

## 7.4 Integration in the overall service oriented system

The integration of the automatic functionality generator in the previously presented system is immediate due to the use of service contract based on current technologies. As such, the generated robot programs and voice recognition grammars can be used in the automatic generation tools shown in chapter 5, and an orchestration program can be used to glue it all together. This example clearly shows the advantages of the declarative nature of the orchestration language presented in chapter 6, which can be generated without any ambiguity.



**Figure 38. Full code generation for PbD work-cell**

Considering that the cell specification described earlier will be used in a programming-by-demonstration system (PdB), the amount of information provided by the cell specification is in this case enough to define the necessary commands. As an example, consider the definition of the robot position “Corner 2”. The voice recognition system should recognize the following commands:

```

Robot Move Linear Corner 2
Robot Move Joint Corner2
Robot Save To Position Corner 2

```

**Listing 26. Voice Commands**

The automatic generation software would thereafter derive the speech recognition grammar presented in Listing 27.

```

<P>
  <P>Robot </P>
  <P>Move</P>
  <L>
    <P>Linear</P>
    <P>Joint</P>
  </L>
  <P>
    <L>
      <P>Corner 2 </P>
      ...
    </L>
  </P>
</P>

```

**Listing 27. Generated speech recognition grammar**

For the robot system, the automatic generation system will generate the equivalent robot code Listing 28.

```
![Action]
PROC MoveLinear(string position)
  GetDataVal position,currPos;
  MoveL currPos, vCurr, fine, tool0;
ENDPROC
```

#### Listing 28. Generated robot code

These last two specifications comply with the software developed before and presented in chapter 5, which can generate services from both robot programs and voice recognition grammars. The orchestration of the generated services is made through the definition of a state-chart presented in Listing 29.

```
<?xml version="1.0"?>
<stateMachine xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" namespace="RoboticsLab" name="CellController"
initialState="Stopped" stateMachineType="Active">
  <state id="Stopped" historyType="None">
    <transition event="urn:schemas-upnp-org:device:VoiceInterface:1;RecognitionService; Robot
MoveLinearCorner2" target="RobotMoving">
      <UPnPAction name="urn:schemas-upnp-org:device:Robot:CellSpecService;MoveLinear" />
      <UPnPArgument name="numToPick" val="urn:schemas-upnp-
org:device:VoiceInterface:1;RecognitionService;CurrPos" />
    </transition>
  </state>
  ...
</state>
```

#### Listing 29. Generated orchestration

The state chart presented is afterwards loaded into the SCXML engine presented in chapter 6, providing a full orchestration of the generated functionality.

### 7.4.1 Demonstration software

The main software is a Java application based in JastAdd-generated files, which in its current version does not have a graphical interface. However, to demonstrate the entire route from cell specification to the UPnP service, a J# application has been developed, making use of the large compatibility of the Java files generated by the JastAdd system (*JRE 1.2*).

The software application presented in Figure 39 parses the cell specification specified in the format defined in the last section, generates a voice recognition grammar for the *Microsoft*

SAPI 5.1 voice recognition system [144], loads it into the engine, generates the respective network service and allows its advertising in the network, thus composing a fully working prototype.

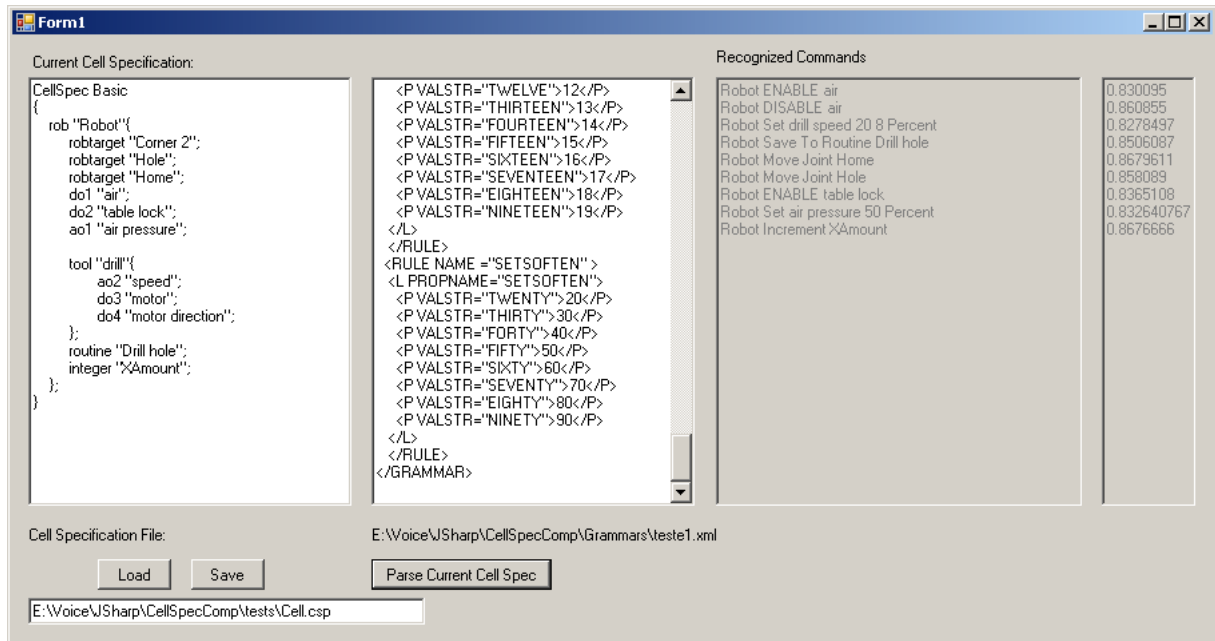


Figure 39. Voice recognition software with automatic generated grammar.

## 7.5 Conclusions

The software developed here shows the great potential for the automatic generation of functionality of a PbD system. This application shows one of the advantages of the system proposed in chapter 4: the **degree of decoupling** from orchestration and execution is such that there is no C# or Java code needed to integrate these devices starting from a simple cell description.



It should be mentioned that, although in this specific case all those software applications could arguably be running in the same CPU, the architecture fully supports networking. Considering for example the integration of a PDA interface like the one described in chapter 6, the distribution needs would immediately arise.

Nevertheless, some extra work with special focus on the information that is still embedded into the system is called for. A prominent example is the extra knowledge needed for the definition of voice commands. Furthermore, a graphical interface for the feedback on the current status of the programming by demonstration systems should be developed.



*Si el gallo gana - dijo la mujer -. Pero si pierde  
No se te ha ocurrido que el gallo puede perder.  
-Es un gallo que no puede perder.*

*El coronel no tiene quien le escriba- Garcia Marquez*

# 8 Conclusions

---

With the challenge of providing easier reconfiguration for SME robotic cells, this thesis has developed an experimental approach to validate a recent enabler in software technologies, device-level SOA (conclusions in section 8.1), and proposed complementary solutions for the enabler to define an SME-programmable robotic work-cell. These solutions are mechanisms for the automatic generation/specification of services (section 8.2), and the specification of a language suitable for the orchestration of device-level SOA-enabled robotic cells (section 8.3). Furthermore, it has explored how a cell specification can be used for the automatic configuration of robotic work-cells (section 8.4).

## 8.1 SOA evaluation

Important conclusions in this work result from the evaluation of two device-level SOA: DSSP and UPnP.

DSSP imports some of the REST principles to the device level. The arrival of DSSP and its related platform (MSRS) defines a very important step in robotics, with the direct involvement of a major software player in the robotics field. DSSP defines a message-based communication

mechanism built around the state of a service, with state-driven events and operations over the service state. Combined with the coordination runtime CCR, DSSP constitutes an excellent platform for the development of highly concurrent systems, such as mobile robots.

In the device level, UPnP represents a group of technologies related to *WebServices* that also include DPWS.

The architectural styles are quite different. DSSP services provide hidden behaviour and public state, which can be accessed through a confined set of operations (GET, SET, UPDATE, etc). This style is very powerful, especially when combined with a specific platform (like the .Net Framework) since it relies on complex data contracts and constructs (arrays, etc) to achieve expressiveness. UPnP provides a combination of RPC interaction style through UPnP actions and a publish/subscribe messaging mechanism through UPnP events. UPnP (and DPWS) rely on standardization efforts to agree on common interfaces, and their development is therefore slower, with limited data contracts and constructs, p.e. On the other hand, these technologies are inherently platform-independent, thus providing more possibilities for business level integration and independent development.

### 8.1.1 Results

This work confirms other research, developed mainly inside the European projects SIRENA [71] and SODA [73], in terms of the advances provided by the use of device-level SOA for industrial automation. Moreover, it concludes that the UPnP/DPWS platform is more adequate to be used in robotic work-cells for SMEs, due to the enhanced expressiveness of the messaging mechanism that allows services to be used both in setup and orchestration, and its increased platform independence.

Looking back, some problems can be listed and some choices questioned. In the SOA evaluation, the choice of UPnP over DPWS can be one of those. Nevertheless, the similarities in the overall architecture allow the results from this thesis to be ported to the DPWS architecture, and the availability of tools allowed the work to focus on other issues, without the need to develop DPWS stack and tools.

### 8.1.2 Future

Considering the robotic work-cell environment, future evolution can determine which of these approaches will be the more successful. If the integration of several devices (e.g. PLCs, vision systems) is going to include sensory and low-level control, the DSSP architectural style will probably take the lead. It is important to note that in an industrial environment, low-level control usually implies real-time behaviour due to obvious safety reasons. The acceptance of the DSSP platform to cope with this type of control is therefore constrained by the development of advanced real-time garbage collection techniques for the .net Framework, like the ones presented in [178] and [179] for Java. In contrast, if the evolution of industrial automation leads to relatively closed systems that deal with almost all of the real-time safety issues within the equipment or proprietary accessories, the extra expressiveness, the platform independence and the better compatibility with current technologies of the UPnP (or probably DPWS) platform should guarantee further acceptance.

In the SME robotic work-cell, where low level control is unlikely to be needed, the extra efforts described in this thesis on the UPnP platform are justified.

## 8.2 Service generation

Service contracts compose a crucial principle in service-oriented systems. Considering an industrial robot, the specification of services based on device features is a limiting misuse, leading to the use of SOA platforms without the necessary supportive SOA tenet. A clear example can be found in the connection developed by ABB to integrate MSRS, the ABB Connect™ [180]. In this specification, there is a definition for an *ABB Controller Service*, which describes access methods to read and write Rapid variables, load modules, run tasks etc. These methods are simply new implementations of old SUN RPCs over a new platform, thus limiting real SOA benefits. When using these platforms, the real functionality description is hidden in the robot program, thus hindering the desirable loose-coupling and a good separation of concerns, computation (execution) and orchestration.

To achieve this separation, this thesis has proposed a mechanism for the specification of contracts that uses robot programs as the source. Robot programs are the key element of robot flexibility, and their use for contract design will feature flexibility also at the interconnection level. The procedural nature of many robot programming languages copes perfectly with the UPnP/DPWS messaging style, with a mixture of state variables defined by robot variables and actions defined by robot methods. The tests made with several users have demonstrated the wanted ease of use and expressiveness in terms of defining functionality. Furthermore, experiments have been made with automatic generation from different types of sources, like voice recognition grammars.

### 8.2.1 Service generation results

The proposed mechanism is a key enabler in the future use of device-level SOA in industrial robotics, due to the interface definition's importance in the success of these architectures. Hopefully, this result can be integrated in future platforms, namely XIRP, due to its robotic origin, thus empowering robot programmers to specify functionality, without the need to exchange the teach pendant with the PC mouse.

### 8.2.2 Future

The specification of a service by users does not invalidate efforts towards new standards, either in research like the Rosta Project [35], or within industry with the XIRP trend [77]. However, these efforts should balance *manufacturer services* with *user-defined services*, in order to provide both automatic configuration between *manufacturer services*, and manual configuration between *user-defined services*. From the evaluation made, the best mechanism to incorporate this balance should be similar to the UPnP Device Control Protocols described in section 3.1.2. They not only provide both types of service specification (user-defined and forum standardized), but also allow the design of hybrid contracts, which, despite complying with a standard and being therefore interoperable, still permit the user to add specific behaviour.

The extension of this concept to a state-driven system like DSSP is not immediate, but there are several possibilities. From our perspective, the best way to define service contracts in state-

based interactions is using the composition of complex data contracts. Some robot languages permit the definition of these structures, like p.e. ABB's Rapid with the keyword RECORD or Fanuc's Karel with the keyword TYPE. A good starting point for future research would be the use of these language constructs to specify evented variables that would later be consumed in the DSSP-framework.

## 8.3 Orchestration of services

One of the most important SOA design principles are the loose-coupling and the good interface definition, mainly due to their importance in orchestration. The SOA vision foresees the application programmer as a client that looks at available services, selects some of them, and establishes orchestrations of services, like workflows p.e., that define process-based composite applications. To achieve this using BPEL4WS, an XML language with limited computation capabilities, the abstracted functionality of the service needs to fit the needs of the consumer. This has been a major challenge in the adoption of BPEL4WS.

As pointed out in section 6.3, the device-level orchestration was an open issue. The use of internet-derived tools on device-level SOA is not appropriate due to the reactive nature of industrial systems, which is not well described by process-based tools like BPEL4WS.

This work proposes an event-driven orchestration language that should be easily accepted by current robot programmers. This language, together with the automatic generation of services starting from robot programs, should give the robot operator the ability to integrate the robot in the network without any programming knowledge (besides the robot's one) and without compromising the flexibility of the system.

### 8.3.1 Results

The results from the tests carried out with an industrial prototype, clearly show that the use of event-driven orchestration techniques, in particular with statecharts, is adequate for the orchestration of robotic work-cells.

Furthermore, the definition of an SCXML-derived language means that another SOA principle has finally been achieved in industrial automation: the orchestration's independence from the programming language. The current implementation of the statechart engine is made in C#, but there are tools available for Java, for instance.

The limitations of the study are: the lack of orchestration tests with a large number of services; the lack of formal evaluation of other event-driven techniques; and the lack of integration of the orchestration into higher layers of the business network.

### 8.3.2 Future

Further studies should be made of this promising concept. Statecharts should be continuously evaluated with a representative number of SME users, with tests of the usability and expressiveness of this formalism. The combination of the event-driven approach with a data-driven approach, like the one presented by IEC61499 [181], should also be considered. Future orchestration techniques will certainly integrate many orchestration schemas, each in a specific area. **Statecharts** and similar in the orchestration of device-level interaction, and **dataflow-driven orchestration techniques** for high-level software integration are examples.

## 8.4 Automatic configuration of work-cells

The paradigm shift induced by the existence of PbD systems will ask for new approaches and the cell specification defined in this work trails a way on how to do it. This teaching method, PdB, shares some of the benefits of traditional (and still widely held) teach-pendant-based systems, namely its immunity to the lack of volumetric accuracy in current robots, but introduces new challenges regarding parameterization, specially of the voice recognition system. This work proposes a language that declaratively addresses this issue, but also uses the capabilities of automatic generation techniques to achieve a new level of automatic configuration: I/O naming, robot code, documentation and orchestration program are generated automatically, thus allowing fast configuration for PbD work-cells.



It has been proved that modern compiler technologies can bridge the gap between ontology-based automatic reasoning systems, and current automation technologies.

In conclusion, this work has shown the importance of the separation of concerns introduced by the orchestration language, which allows the generation of reusable maintainable code without extra effort. This means future developments in automatic reasoning systems should carefully consider the use of device-level SOA and a separate orchestration language.

### 8.4.1 Future

This work opens the door for further studies of a more complete cell specification, which can provide manufacturers with a valuable tool during the parameterization of a system, especially if this specification is standard.

In addition, the graphical interface's generation and the speech feedback should be integrated in the whole system.

## 8.5 Final remarks

This thesis defines a set of principles for the design of future device-level SOA-related tools. These principles may need to be further evaluated, but they reflect a careful look into the robotic work-cell reality and its future challenges, namely PbD. The work has combined results from the industrial prototype developed in the laboratory, and knowledge gained by the author through his laboratory's industrial experience while developing industrial systems. When evaluating needs, it is always positive to live close to them.

It is therefore the author's belief that, with a robotic work-cell that is service-oriented, and with service interfaces designed both by the manufacturer and the cell user, but always reflecting high-level functionality orchestrated with a simple statechart-like type of language, future robots will be more-widely accepted in SMEs. This will enhance a major trend in our society: to provide better working conditions to a broader range of population.



# Bibliography

---

- [1] "European Comission - Growth and Jobs - Key Documents." [Online], 2006. [ec.europa.eu/growthandjobs/key/index\\_en.htm](http://ec.europa.eu/growthandjobs/key/index_en.htm).
- [2] IFR, "World Robotics 2006. Statistics, Market Analysis, Forecasts, Case Studies and Profitability of Robot Investments." [Online], 2006. [www.worldrobotics-online.org](http://www.worldrobotics-online.org).
- [3] K. Nilsson, R. Johansson, A. Robertsson, R. Bischoff, T. Brogardh, and M. Hägele, "Productive Robots and the SMERobot Project," *Third Swedish Workshop on Autonomous Robotics*, 2005.
- [4] J. Decotignie, "Ethernet-Based Real-Time and Industrial Communications," *Proceedings of the IEEE*, vol. 93, 2005, pp. 1102-1117.
- [5] Xinggang Fan, Zhi Wang, and Youxian Sun, "How to guarantee factory communication with switched Ethernet: survey of its emerging technology," *IECON 02 [Industrial Electronics Society, IEEE 2002 28th Annual Conference of the]*, 2002, pp. 2525-2530 vol.3.
- [6] K.C. Lee and S. Lee, "Performance evaluation of switched ethernet for real-time industrial communications," *Comput. Stand. Interfaces*, vol. 24, 2002, pp. 411-423.
- [7] P. Pedeiros, Leite, R., and Almeida, L., "Characterizing the real time behaviour of prioritized switched Ethernet.," Viena: 2003.
- [8] *IEEE Standards for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges*, IEEE Computer Society, 2005.
- [9] *IEEE Standards for Local and Metropolitan Area Networks:Virtual Bridged Local Area Networks*, IEEE Computer Society, 2005.
- [10] *IEEE Standards for Local and Metropolitan Area Networks:LAN Layer 2 QoS/CoS Protocol for Traffic Prioritization*, IEEE Computer Society, 2005.

- [11] Baek-Young Choi, Sejun Song, N. Birch, and Jim Huang, "Probabilistic approach to switched Ethernet for real-time control applications," *Real-Time Computing Systems and Applications, 2000. Proceedings. Seventh International Conference on*, 2000, pp. 384-388.
- [12] Hoai Hoang, M. Jonsson, U. Hagstrom, and A. Kallerdahl, "Switched real-time ethernet with earliest deadline first scheduling protocols and traffic handling," *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, 2002, pp. 94-99.
- [13] X. Fan and M. Jonsson, "Efficient support for high traffic-volumes of short-message real-time communication using an active Ethernet switch," *Proc. of 10th International Conference on Real-Time and Embedded computing Systems*, pp. 517-533.
- [14] A. Koubaa and Y.-Q. Song, "Evaluation and improvement of response time bounds for real-time applications under non-pre-emptive Fixed Priority Scheduling," *International Journal of Production Research*, vol. 42, Jul. 2004, pp. 2899-2913.
- [15] J. Jasperneite, P. Neumann, M. Theis, and K. Watson, "Deterministic real-time communication with switched Ethernet," *Factory Communication Systems, 2002. 4th IEEE International Workshop on*, 2002, pp. 11-18.
- [16] Xing Fan and M. Jonsson, "Guaranteed real-time services over standard switched Ethernet," *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, 2005, p. 3 pp.
- [17] "IEC Web Store | Publication Detail > IEC 61784-3-3 Ed. 1.0 English" [Online]. <http://webstore.iec.ch/webstore/webstore.nsf/artnum/038818>.
- [18] M. Felser, "Real-Time Ethernet - Industry Prospective," *Proceedings of the IEEE*, vol. 93, 2005, pp. 1118-1129.
- [19] J. Jasperneite, M. Schumacher, and K. Weber, "Limits of increasing the performance of Industrial Ethernet protocols," *Emerging Technologies & Factory Automation, 2007. ETFA. IEEE Conference on*, 2007, pp. 17-24.
- [20] P. Neumann, "Communication in industrial automation--What is going on?," *Control Engineering Practice*, vol. 15, Nov. 2007, pp. 1332-1347.
- [21] "ODVA : EtherNet/IP" [Online]. <http://www.odva.org/default.aspx?tabid=67>.
- [22] "PROFINET" [Online]. <http://www.profibus.com/pn/>.
- [23] "Modbus-IDA Specifications and Implementation Guides" [Online]. <http://www.modbus.org/specs.php>.
- [24] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems- IEEE 1558*, IEEE, 2002.

- [25] "EtherCAT Technology Group" [Online]. <http://www.ethercat.org/>.
- [26] "ETHERNET Powerlink - Home" [Online]. <http://www.ethernet-powerlink.org/>.
- [27] "The Industrial Ethernet Book - Articles: Modbus, real time, determinism and reality in plant floor control" [Online]. <http://ethernet.industrial-networking.com/articles/articledisplay.asp?id=1672>.
- [28] A. Kazi, J. Bunsendal, D. Haag, R. Baum, and R. Bischoff, "Next Generation Teach Pendants for Industrial Robots" [Online], *Advances in Human-Robot Interaction*, 2005. <http://www.springerlink.com/content/ac15e5b777e373d5>.
- [29] ABB, *ABB IRC5 Application Manual Force Control Machining FC*, Vasteras: 2007.
- [30] A. Blomdell, G. Bolmsjo, T. Brogardh, P. Cederberg, M. Isaksson, R. Johansson, M. Haage, K. Nilsson, M. Olsson, T. Olsson, A. Robertsson, and Jianjun Wang, "Extending an industrial robot controller: implementation and applications of a fast open sensor interface," *Robotics & Automation Magazine, IEEE*, vol. 12, 2005, pp. 85-94.
- [31] Pires, J. N., Veiga, G., and Araujo, R., "Programming-by-demonstration in the coworker scenario for SMEs," *Industrial Robot, Emerald*, vol. 36, 2008.
- [32] Zengxi, Pan and Zhang, Hui, "Robotic machining from programming to process control: a complete solution by force control," *Industrial Robot: An International Journal, Emerald*, vol. 35.
- [33] M. Naumann, K. Wegener, R. Schraft, and L. Lachello, "Robot cell integration by means of application-P'n'P," 2006.
- [34] "SIARAS FP6 PROGRAMME PROJECT SUMMARY" [Online]. <http://www.siaras.org/project.html>.
- [35] "RoSta: Project" [Online]. <http://www.robot-standards.org/index.php?id=8>.
- [36] "OWL Web Ontology Language Overview" [Online]. <http://www.w3.org/TR/owl-features/>.
- [37] "What is Protégé-OWL?" [Online]. <http://protege.stanford.edu/overview/protege-owl.html>.
- [38] L. Lin, M. Wakabayashi, and S. Adiga, "Object-oriented modeling and implementation of control software for a robotic flexible manufacturing cell," *Robotics*, vol. 11, 1994, pp. 1-12.
- [39] J. Pires and J. Sa da Costa, "Object-oriented and distributed approach for programming robotic manufacturing cells," *Robotics and Computer-Integrated Manufacturing*, vol. 16, Feb. 2000, pp. 29-42.

- [40] "Motoman - MotoCom SDK" [Online]. <http://www.motoman.com/products/software/program/MotoCom%20SDK.php>.
- [41] "ABB WebWare Server - Software Products (Robotics)" [Online]. <http://www.abb.com/product/seitp327/12e18c81002601cac1256f2b003b638e.aspx>.
- [42] P. Kopacek, G. Kronreif, and R. Probst, "A modular control system for flexible robotized manufacturing cells," *Robotica*, vol. 17, 1999, pp. 23-32.
- [43] M. Bruccoleri, "Reconfigurable control of robotized manufacturing cells," *Robot. Comput.-Integr. Manuf.*, vol. 23, 2007, pp. 94-106.
- [44] Ling Gou, P. Luh, and Y. Kyoya, "Holonc manufacturing scheduling: architecture, cooperation mechanism, and implementation," *Advanced Intelligent Mechatronics '97. Final Program and Abstracts., IEEE/ASME International Conference on*, 1997, p. 39.
- [45] G. Veiga and R. Cancela, "Advanced HMI for robot programming: An industrial application for the ceramic industry.," *Proc. International Symposium on Robotics.*, Barcelona: 2008.
- [46] J. Pires and S. Paulo, "High efficient robotic de-palletizing system for the non-flat ceramic industry," *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, 2003, pp. 3529-3534 vol.3.
- [47] G. Lopes, "Processo de Lixagem Robotizado para a Indústria das Cutelarias," 2008.
- [48] J. Pires, T. Godinho, K. Nilsson, M. Haage, and C. Meyer, "Programming industrial robots using advanced input-output devices: test-case example using a CAD package and a digital pen based on the Anoto technology," *International Journal of Online Engineering*, vol. 3, 2007.
- [49] E.W. Dijkstra, "Chapter I: Notes on structured programming," *Structured programming*, Academic Press Ltd., 1972, pp. 1-82.
- [50] O. Dahl and K. Nygaard, "SIMULA: an ALGOL-based simulation language," *Commun. ACM*, vol. 9, 1966, pp. 671-678.
- [51] M. Henning, "The rise and fall of CORBA," *Queue*, vol. 4, 2006, pp. 28-34.
- [52] "Mono" [Online], 2008. <http://www.mono-project.com/Mono>About>.
- [53] "World Wide Web Consortium - Web Standards" [Online]. <http://www.w3.org/>.
- [54] "OASIS: Advancing open standards for the global information society" [Online]. <http://www.oasis-open.org/home/index.php>.
- [55] "OASIS - Committees - OASIS UDDI Specifications TC" [Online]. <http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm>.
- [56] "Web Service Definition Language (WSDL)" [Online]. <http://www.w3.org/TR/wsdl>.

- [57] "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)" [Online]. <http://www.w3.org/TR/2007/REC-soap12-part1-20070427/>.
- [58] W. Vogels, "Web services are not distributed objects," *Internet Computing, IEEE*, vol. 7, 2003, pp. 59-66.
- [59] R.T. Fielding, "Architectural styles and the design of network-based software architectures," 2000.
- [60] R.T. Fielding and R.N. Taylor, "Principled design of the modern Web architecture," *Proceedings of the 22nd international conference on Software engineering*, Limerick, Ireland: ACM, 2000, pp. 407-416.
- [61] D. Harel and A. Pnueli, "On the development of reactive systems," *Logics and models of concurrent systems*, Springer-Verlag New York, Inc., 1985, pp. 477-498.
- [62] A. Pnueli, "Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends," *Current trends in concurrency. Overviews and tutorials*, Springer-Verlag New York, Inc., 1986, pp. 510-584.
- [63] E. Sekerinski, "Graphical Design of Reactive Systems," *Proceedings of the Second International B Conference on Recent Advances in the Development and Use of the B Method*, Springer-Verlag, 1998, pp. 182-197.
- [64] A. Bhattacharjee, S. Dhodapkar, S. Seshia, and R. Shyamasundar, "A Graphical Environment for the Specification and Verification of Reactive Systems" [Online], *Computer Safety, Reliability and Security*, 1999. [http://dx.doi.org/10.1007/3-540-48249-0\\_37](http://dx.doi.org/10.1007/3-540-48249-0_37).
- [65] D. Lea, S. Vinoski, and W. Vogels, "Guest Editors' Introduction: Asynchronous Middleware and Services," *IEEE Internet Computing*, vol. 10, 2006, pp. 14-17.
- [66] P.T. Eugster, P.A. Felber, R. Guerraoui, and A.-. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, 2003, p. 114—131.
- [67] L. Aldred, W.M. van der Aalst, M. Dumas, and A.H. ter Hofstede, "On the Notion of Coupling in Communication Middleware" [Online], *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, 2005. [http://dx.doi.org/10.1007/11575801\\_6](http://dx.doi.org/10.1007/11575801_6).
- [68] "UPnP forum" [Online], 2000. [www.upnp.org](http://www.upnp.org).
- [69] "Main Page - Jini.org" [Online], 1999. [http://www.jini.org/wiki/Main\\_Page](http://www.jini.org/wiki/Main_Page).
- [70] "OSGi Alliance | Main / OSGi Alliance" [Online]. <http://www.osgi.org/Main/HomePage>.
- [71] "Home - Sirena 2003 - 2005" [Online], 2005. <http://www.sirena-itea.org/Sirena/Home.htm>.

- [72] "Home - Socrates - 2006" [Online], 2006. <http://www.socrates.eu/Home/default.html>.
- [73] "Home - Soda - 2006" [Online], 2006. <http://www.soda-itea.org/Home/default.html>.
- [74] Nielsen, H and Chrynsanthakopoulos, G, "Decentralized Software Services Protocol – DSSP/1.0" [Online], 2006. <http://purl.org/msrs/dssp.pdf>.
- [75] I. Delamer and J. Lastra, "Loosely-coupled Automation Systems using Device-level SOA," *Industrial Informatics, 2007 5th IEEE International Conference on*, 2007, pp. 743-748.
- [76] F. Jammes and H. Smit, "Service-oriented paradigms in industrial automation," *Industrial Informatics, IEEE Transactions on*, vol. 1, 2005, pp. 62-70.
- [77] D. Fan and J. Unger, "Towards a standardized and extensible mechanism for robot device integration," *Informatics in Control, Automation and Robotics*, Madeira: 2008.
- [78] Leitner, S and Mahnke, W, "OPC UA – Service-oriented Architecture for Industrial Applications," Ladenburg: 2006.
- [79] "UPnP™ Standards" [Online]. <http://www.upnp.org/standardizeddcps/>.
- [80] J. Schlimmer, *UPnP PlayCD:1 Sample Service Template For UPnP™ Version 1.0*, UPnP Forum, 2001.
- [81] "Cover Pages: Microsoft Releases Devices Profile for Web Services Specification." [Online]. <http://xml.coverpages.org/ni2004-05-04-a.html>.
- [82] "UPnP Forum and DPWS Standardization Status" [Online]. [http://74.125.39.104/search?q=cache:Ec9J0ZGJJE0J:download.microsoft.com/download/f/0/5/f05a42ce-575b-4c60-82d6-208d3754b2d6/UPnP\\_DPWS\\_RS08.pptx+UPnP+dpws&hl=en&ct=clnk&cd=2](http://74.125.39.104/search?q=cache:Ec9J0ZGJJE0J:download.microsoft.com/download/f/0/5/f05a42ce-575b-4c60-82d6-208d3754b2d6/UPnP_DPWS_RS08.pptx+UPnP+dpws&hl=en&ct=clnk&cd=2).
- [83] "EUREKA | A Europe-wide Network for Market-Oriented Industrial R&D and Innovation" [Online]. <http://www.eureka.be/inaction/viewSuccessStory.do?docid=2631219>.
- [84] A. Bobek, E. Zeeb, H. Bohn, F. Golatowski, and D. Timmermann, "Device and service templates for the Devices Profile for Web Services," *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, 2008, pp. 797-801.
- [85] "DSS Service Components" [Online]. <http://msdn.microsoft.com/en-us/library/bb483067.aspx>.
- [86] J. Jackson, "Microsoft robotics studio: A technical introduction," *Robotics & Automation Magazine, IEEE*, vol. 14, 2007, pp. 82-87.
- [87] G. Chrysanthakopoulos and S. Singh, "An Asynchronous messaging library for C#," *Proc. Annual Conf. Object-Oriented Programming, Systems, Languages and Applications*, Anaheim: 2003.



- [88] "CCR Coordination Primitives" [Online]. <http://msdn.microsoft.com/en-us/library/bb648749.aspx>.
- [89] W. Lu, T. Gunarathne, and D. Gannon, "Developing a concurrent service orchestration engine in ccr," *Proceedings of the 1st international workshop on Multicore software engineering*, Leipzig, Germany: ACM, 2008, pp. 61-68.
- [90] "NVIDIA PhysX" [Online]. [http://www.nvidia.com/object/nvidia\\_physx.html](http://www.nvidia.com/object/nvidia_physx.html).
- [91] J. Pires, R. Araújo, and T. Godinho, "Controlo e Monitorização de Células Robotizadas de Produção, Desenvolvimento de uma Aplicação utilizando WebCam, PC, PLC e sockets TCP/IP," *Robótica*, 2006.
- [92] "Intel® Software for UPnP\* Technology" [Online]. <http://www.intel.com/cd/ids/developer/asmo-na/eng/downloads/upnp/index.htm>.
- [93] ABB, *ABB - RAPID Instructions, Functions and Data types*, ABB, 2008.
- [94] "Universal Plug and Play" [Online]. <http://www.microsoft.com/whdc/device/media/upnp.mspix>.
- [95] "SourceForge.net: Open Computer Vision Library" [Online]. <http://sourceforge.net/projects/opencvlibrary/>.
- [96] "Siemens AG- UPnP" [Online]. <http://www.plugin-play-technologies.com/>.
- [97] "Windows Mobile 6 SDK" [Online]. <http://www.microsoft.com/downloads/details.aspx?FamilyID=06111A3A-A651-4745-88EF-3D48091A390B&displaylang=en>.
- [98] "Overview of the .NET Compact Framework" [Online]. <http://msdn.microsoft.com/en-us/library/w6ah6cw1.aspx>.
- [99] "Productive Robotics - Homepage" [Online]. <http://www.robot.lth.se/java/>.
- [100] F. UPnP, "Device Security and Security Console V 1.0" [Online]. <http://www.upnp.org/standardizeddcp/security.asp>.
- [101] R. Wareham, "Ladder diagram and sequential function chart languages in programmable controllers," *Programmable Control and Automation Technology Conference and Exhibition, 1988. Conference Proceedings., Fourth Annual Canadian*, 1988, pp. 12A-14/1-4.
- [102] IEC, "IEC 61131-3, 2nd Ed. Programmable Controllers – Programming Languages," 2003.

- [103] I. Delamer and J. Lastra, "Ontology Modeling of Assembly Processes and Systems using Semantic Web Services," *Industrial Informatics, 2006 IEEE International Conference on*, 2006, pp. 611-617.
- [104] J. Malec, A. Nilsson, K. Nilsson, and S. Nowaczyk, "Knowledge-Based Reconfiguration of Automation Systems," *Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on*, 2007, pp. 170-175.
- [105] J. Lastra and M. Delamer, "Semantic web services in factory automation: fundamental insights and research roadmap," *Industrial Informatics, IEEE Transactions on*, vol. 2, 2006, pp. 1-11.
- [106] D. Gelernter and N. Carriero, "Coordination languages and their significance.," *Commun. ACM*, vol. 35, 1992, pp. 97-107.
- [107] N. Carriero and D. Gelernter, "Linda in context," *Commun. ACM*, vol. 32, 1989, pp. 444-458.
- [108] J. Browne, S. Hyder, J. Dongarra, K. Moore, and P. Newton, "Visual programming and debugging for parallel computing," *Parallel & Distributed Technology: Systems & Applications, IEEE*, vol. 3, 1995, pp. 75-83.
- [109] U. Borgolte, "The New German Standard Robot Programming Language IRL," 1994.
- [110] Motoman, *Motoman NX100 INFORM programming manual* [Online], 2006. <http://www.google.pt/firefox?client=firefox-a&rls=org.mozilla:en-GB:official>.
- [111] *FANUC Robot series Handling Tool*, 2008.
- [112] *FANUC Robotics SYSTEM R-J3iB Controller KAREL Reference Manual*, 2002.
- [113] *KUKA Documentation V5.x - Advance Programming*, 2006.
- [114] Comau Robotics, *Comau PDL2 Programming Language Manual System Software*, 2005.
- [115] ABB, *ABB - RAPID kernel*, ABB, 2008.
- [116] C. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing* [Online], {The MIT Press}, 1999. <http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0262133601>.
- [117] C. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, Jul. 1948, pp. 423, 379.
- [118] S. Chen, B. Kingsbury, Lidia Mangu, D. Povey, G. Saon, H. Soltau, and G. Zweig, "Advances in speech transcription at IBM under the DARPA EARS program," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 14, 2006, pp. 1596-1608.

- [119] "Effective Affordable Reusable Speech-to-Text" [Online]. <https://ssli.ee.washington.edu/projects/EARS.html>.
- [120] L. Deng and X. Huang, "Challenges in adopting speech recognition," *Commun. ACM*, vol. 47, 2004, pp. 69-75.
- [121] R.P. Lippmann, "Speech recognition by machines and humans," *Speech Commun.*, vol. 22, 1997, pp. 1-15.
- [122] L. Deng and D. O'Shaughnessy, *Speech Processing: A Dynamic and Optimization-oriented Approach*, Marcel Dekker, 2003.
- [123] J. Allen, "How do humans process and recognize speech?," *Speech and Audio Processing, IEEE Transactions on*, vol. 2, 1994, pp. 567-577.
- [124] J. Allen, "From Lord Rayleigh to Shannon: How do we decode speech?," *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, 2002.
- [125] M. Jeong, "Using higher-level linguistic knowledge for speech recognition error correction in a spoken q/a dialog," *In: Proceedings of the HLT-NAACL special workshop on Higher-Level Linguistic Information for Speech Processing*, 2004, p. 48—55.
- [126] C. Che, Q. Lin, J. Pearson, B.D. Vries, and J. Flanagan, "Microphone arrays and neural networks for robust speech recognition," *Proceedings of the workshop on Human Language Technology*, Plainsboro, NJ: Association for Computational Linguistics, 1994, pp. 342-347.
- [127] I. Himawan, S. Sridharan, and I. McCowan, "Dealing with uncertainty in microphone placement in a microphone array speech recognition system," *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, 2008, pp. 1565-1568.
- [128] "SPEECHMAGIC - Home | SpeechMagic, speech recognition, voice recognition, document creation, customer stories, news center" [Online]. <http://www.speechmagic.com/>.
- [129] "Voice Extensible Markup Language (VoiceXML) Version 2.0" [Online]. <http://www.w3.org/TR/voicexml20/>.
- [130] "Speech Recognition Grammar Specification Version 1.0" [Online]. <http://www.w3.org/TR/speech-grammar/>.
- [131] "Semantic Interpretation for Speech Recognition (SISR) Version 1.0" [Online]. <http://www.w3.org/TR/semantic-interpretation/>.
- [132] "Speech Synthesis Markup Language (SSML) Version 1.0" [Online]. <http://www.w3.org/TR/speech-synthesis/>.

- [133] "Voice Browser Call Control: CCXML Version 1.0" [Online]. <http://www.w3.org/TR/ccxml/>.
- [134] J. Barnett, M. Bodell, D. Burnett, J. Carter, and R. Hosn, "State Chart XML (SCXML): State Machine Notation for Control Abstraction" [Online]. <http://www.w3.org/TR/2007/WD-scxml-20070221/>.
- [135] J. Foster and S. Bryson, "Voice recognition for the IBM 7535 robot," *Southeastcon '89. Proceedings. Energy and Information Technologies in the Southeast., IEEE*, 1989, pp. 759-764 vol.2.
- [136] D. Mital and G. Leng, "A voice-activated robot with artificial intelligence," *Industrial Electronics, Control and Instrumentation, 1991. Proceedings. IECON '91., 1991 International Conference on*, 1991, pp. 904-909 vol.2.
- [137] J. Pires, "Robot-by-voice: experiments on commanding an industrial robot using the human voice," *Industrial Robot: An International Journal*, vol. 32, pp. 505-511.
- [138] "Voxware: Voice Software for Voice-Based Warehousing" [Online]. <http://www.voxware.com/>.
- [139] J. Hopcroft, R. Motwani, and J. Ullman, *Introduction to Automata Theory, Languages, and Computation (2nd Edition)* [Online], Addison Wesley, 2000. <http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0201441241>.
- [140] "Standard ECMA-262" [Online]. <http://www.ecma-international.org/publications/standards/Ecma-262.htm>.
- [141] "Text grammar format (SAPI 5.3)" [Online]. [http://msdn.microsoft.com/en-us/library/ms723635\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms723635(VS.85).aspx).
- [142] "Grammar Format Specification" [Online]. <http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/index.html>.
- [143] "Understanding GSL" [Online]. <https://studio.tellme.com/grammars/gsl/>.
- [144] "SAPI" [Online]. <http://research.microsoft.com/srg/sapi.aspx>.
- [145] "XAML Overview" [Online]. <http://msdn.microsoft.com/en-us/library/ms752059.aspx>.
- [146] "JavaFX" [Online]. <http://www.sun.com/software/javafx/>.
- [147] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services - Concepts, Architectures and Applications*, Springer, 2003.
- [148] S. Thatte, "Web Services for Business Process Design," 2001.

- [149] "Web Services Flow Language."
- [150] "Web Services Conversation Language (WSCL) 1.0" [Online]. <http://www.w3.org/TR/2002/NOTE-wscl10-20020314/>.
- [151] "OASIS Web Services Business Process Execution Language (WSBP EL) TC" [Online]. [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsbpel](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel).
- [152] S.[. Dustdar and W.[. Schreiner, "A survey on web services composition," *International Journal of Web and Grid Services*, vol. 1, Aug. 2005, pp. 1-30.
- [153] B. Srivastava, "Web service composition - current solutions and open problems," *In: ICAPS 2003 Workshop on Planning for Web Services*, 2003, p. 28—35.
- [154] N. Milanovic and M. Malek, "Current solutions for Web service composition," *Internet Computing, IEEE*, vol. 8, 2004, pp. 51-59.
- [155] E. Motta, J. Domingue, L. Cabral, and M. Gaspari, "IRS-II: A Framework and Infrastructure for Semantic Web Services," Springer-Verlag, 2003, p. 306—318.
- [156] E.F. Jos de Bruijn, "Logical reconstruction of RDF and ontology languages," *Principles and Practice of Semantic Web Reasoning: Second International Workshop, PPSWR 2005*, 2005.
- [157] T.R.G. Green and M. Petre, "Usability Analysis of Visual Programming Environments: A [']Cognitive Dimensions' Framework," *Journal of Visual Languages & Computing*, vol. 7, Jun. 1996, pp. 131-174.
- [158] K.N. WHITLEY and A.F. BLACKWELL, "Visual Programming in the Wild: A Survey of LabVIEW Programmers," *Journal of Visual Languages & Computing*, vol. 12, Aug. 2001, pp. 435-472.
- [159] R. Bischoff, A. Kazi, and M. Seyfarth, "The MORPHA style guide for icon-based programming," *Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on*, 2002, pp. 482-487.
- [160] F. Jammes, H. Smit, J. Lastra, and I. Delamer, "Orchestration of service-oriented manufacturing processes," *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, 2005, pp. 8 pp.-624.
- [161] A. Colombo, F. Jammes, H. Smit, R. Harrison, J. Lastra, and I. Delamer, "Service-oriented architectures for collaborative automation," *Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE*, 2005, p. 6 pp.
- [162] A. Mensch, F. Depeisses, and H. Smit, *Technical Framework Description SODA* [Online], 2007. <http://www.soda-itea.org/Documents/objects/file1176731057.06>.

- [163] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Program.*, vol. 8, 1987, pp. 231-274.
- [164] "SCXML - Commons SCXML" [Online]. <http://commons.apache.org/scxml/>.
- [165] M. Samek, *Practical statecharts in C/C++: Quantum programming for embedded systems*, CMP Publications, Inc., 2002.
- [166] J. Hofschule, "Lead Through Programming" [Online]. [http://www.smerobot.org/09\\_calendar/20071108\\_PressetagABB-Ladenburg\\_LeadThrough.pdf](http://www.smerobot.org/09_calendar/20071108_PressetagABB-Ladenburg_LeadThrough.pdf).
- [167] "Visual Components - Configuration, Visualisation, Simulation, Optimization" [Online]. <http://www.visualcomponents.com/index.php?id=141>.
- [168] "Dassault Systemes: DELMIA - PLM Solutions - Digital Manufacturing and Production" [Online]. <http://www.3ds.com/products/delmia/welcome/>.
- [169] R. Johansson, D. Williams, A. Berglund, and P. Nuges, *Carsim: A System to Visualize Written Road Accident Reports as Animated 3D Scenes*.
- [170] G. Hedin and E. Magnusson, "JastAdd--an aspect-oriented compiler construction system," *Science of Computer Programming*, vol. 47, Apr. 2003, pp. 37-58.
- [171] T. Ekman and G. Hedin, "The jastadd extensible java compiler," *SIGPLAN Not.*, vol. 42, 2007, pp. 1-18.
- [172] "javacc: JavaCC Home" [Online]. <https://javacc.dev.java.net/>.
- [173] A. Appel, *Modern Compiler implementation in Java*, Cambridge University Press, 2002.
- [174] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, Reading, MA, 1995.
- [175] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W.G. Griswold, "An Overview of AspectJ," *Proceedings of the 15th European Conference on Object-Oriented Programming*, Springer-Verlag, 2001, pp. 327-353.
- [176] H.H. Vogt, S.D. Swierstra, and M.F. Kuiper, "Higher order attribute grammars," *Proceedings of the ACM SIGPLAN 1989 Conference on Programming language design and implementation*, Portland, Oregon, United States: ACM, 1989, pp. 131-145.
- [177] T. Ekman and G. Hedin, "Rewritable Reference Attributed Grammars" [Online], *Lecture Notes in Computer Science*, 2004, pp. 171, 147. <http://www.metapress.com/link.asp?id=JEL9AU2PA1F766U7>.
- [178] "Programming in Real-Time Specification for Java (RTSJ): A Conversation with Distinguished Engineer Greg Bollella" [Online]. [http://java.sun.com/developer/technicalArticles/Interviews/Bollella\\_qa2.html](http://java.sun.com/developer/technicalArticles/Interviews/Bollella_qa2.html).

- [179] B. Magnusson and R. Henriksson, "Garbage collection for hard real-time systems," *Object-Orientation in Operating Systems, 1995., Fourth International Workshop on*, 1995, pp. 60-63.
- [180] ABB, "ABB Connect - Users Guide" [Online].  
<http://www.abb.com/product/ap/seitp327/049ffeb82687791ec12574b9004958cf.aspx>.
- [181] IEC, "IEC 61499, 2nd Ed. Programmable Controllers – Programming Languages," 2003.
- [182] Y. Maeda, H. Kikuchi, H. Izawa, H. Ogawa, M. Sugi, and T. Arai, "'Plug Produce" functions for an easily reconfigurable robotic assembly cell," *Assembly Automation*, vol. 27, Aug. 2007, pp. 253-260.





# Appendix A - Abbreviations

---

ASR	Automatic Speech Recognition
AST	Abstract Syntax Tree
ATL	Active Template Library
BPEL4WS	Business Process Execution Language for Web Services
CAM	Computer Aided Manufacturing
CCM	Corba Component Model
CCR	Concurrency RunTime
CFG	Context-Free Grammar
CNC	Computer Numerical Control
CRUD	Create Retrieve Update Delete
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DAML-S	DARPA Agent Markup Language for Webservices
DARPA	Defense Advanced Research Projects Agency
DHCP	Dynamic host configuration protocol
DPWS	Device Profile for Web Services
DSL	domain specific languages
DSSP	Decentralized Software Services Protocol
EARS	Effective Affordable Reusable Speech-to-Text
EJB	Enterprise Java Beans
EU	European Union
GHMI	Graphical Human Machine Interface
HMM	Hidden Markov Models
J2EE	Java 2 Enterprise Edition
JTA	Java Transaction API
LSEs	Large Scaled Enterprises
MAC	medium access control
MOM	Message-oriented middleware
MSRS	Microsoft Robotics Studio
MTS	Microsoft Transaction Server
MVLP	Microsoft Visual Language Programming
NTAs	Non-terminal attributes
OASIS	Organization for the Advancement of Structured Information Standards
OPC-UA	OLE for Process Control – Unified Architecture
OSGI	Open Services Gateway initiative
OWL	Ontology web Language

PbD	Programming-by-Demonstration
PC	Personal Computer
PDA	Personal Digital Assistant
PLC	Programmable Logic Controller
RAGs	Reference Attributed Grammars
REST	Representational State Transfer
RPC	Remote Procedural Calls
SAWSDL	Semantic Annotations for Web Services Description Language
SCADA	Supervisory Control and Data Acquisition
SCXML	State Chart XML
SIRENA	Service Infrastructure for Real-time Embedded Networked Applications
SME	Small Medium Enterprises
SOA	Service Oriented Architectures
SSDP	Simple Service Discovery Protocol
UDDI	Universal Description an Discovery Integration
UPnPDCP	UPnP Device Control Protocols
URIs	Universal Resource Identifiers
USB	Universal Serial Bus
USN	Unique Identification Number
TTS	text to speech
VLANs	Virtual Local Area Networks
WCF	Windows Communication Foundation
WSDL	Web Service Description Language
WSIT	Web Services Interoperability Technologies
W3C	World Wide Web Consortium
WS	Web Services
XAML	eXtensible Application Markup Language
XIRP	XML-based Interface for Robots and Peripherals
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language

## Appendix B – Video List

---

1. Video – Simple Orchestration engine  
<http://robotics.dem.uc.pt/IMS07.wmv>
2. Video – Service Generation from robot programs -  
<http://robotics.dem.uc.pt/RapidUPnP2.wmv>
3. Video – Service Generation from robot programs  
<http://robotics.dem.uc.pt/RapidUPnP.wmv>.
4. Video – Statechart orchestration  
<http://robotics.dem.uc.pt/StateChart.wmv>
5. Video – Statechart orchestration  
<http://robotics.dem.uc.pt/StateChart2.wmv>
6. Video – WiiMote with Statecharts  
<http://robotics.dem.uc.pt/WiiStateChart.wmv>

